VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

VoIP bezpečnostní prvek na bázi SIP Proxy Kamailio a Asterisk PBX
Kamailio and Asterisk based VoIP Security Solution

2020 Hai Dat Pham

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání diplomové práce

Student: **Bc. Hai Dat Pham**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2601T013 Telekomunikační technika

Téma: VoIP bezpečnostní prvek na bázi SIP Proxy Kamailio a Asterisk PBX
Kamailio and Asterisk based VoIP Security Solution

Jazyk vypracování: čeština

Zásady pro vypracování:

Nasazení bezpečnostních politik je dnes nutností pro všechny informační systémy připojené k síti Internet. U VoIP systémů je tato nutnost dále akcentovaná faktem, že jejich napadení může vést k vysokým přímým i nepřímým finančním ztrátám. Využití klasických firewallů neposkytuje ochranu proti pokročilým útokům na aplikační vrstvě, pročež je vhodné zabezpečení VoIP infrastruktury rozšířit o prvky typu SBC (Session Border Controller). Realizace bezpečnostních politik na bázi open-source SIP serverů je tak velmi aktuálním tématem.

Zásady pro vypracování:
1. Student provede rešerši architektury a fungování SIP PBX Asterisk a SIP Proxy Kamailio s důrazem na možnosti využití těchto SW v oblasti bezpečnosti.
2. Student navrhne a realizuje základní bezpečnostní politiky s využitím PBX Asterisk a SIP Proxy Kamailio a identifikuje klíčové vlastnosti obou SW.
3. Student otestuje navržené řešení a provede analýzu flexibility a výkonnosti navrženého řešení.
4. Student zhodnotí základní vlastnosti a výhody navrženého řešení.

Seznam doporučené odborné literatury:

[1] Bryant, R. Madsen, L. Asterisk: *The Definitive Guide*. ISBN 1449332420.
[2] Johnston, A.B. SIP:*Understanding the Session Initiation Protocol*. ISBN 978-1608078639.
[3] Goncalves, F. *Building Telephony Systems with OpenSIPS*. ISBN: 978-1785280610.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Rozhon, Ph.D.**

Datum zadání: 01.09.2018
Datum odevzdání: 30.04.2020

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky 1D16 OSTRAVA

prof. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě, dne 14.5. 2020

.....
Podpis

Poděkování

Na tomto místě bych rád poděkoval vedoucí mé diplomové práce Ing. Janu Rozhonovi, Ph.D. za odbornou pomoc a konzultaci, kterou mi poskytl při zpracovávání mé diplomové práce. Rád bych také poděkoval také své rodině, a hlavně mé sestře, která mě při vytváření této práce podpořila, a bez její pomoci by nebylo možné práci dokončit.

Abstrakt

Nasazení bezpečnostních politik je dnes nezbytné pro všechny informační systémy připojené do veřejné sítě Internet. Výjimkou nejsou ani VoIP systémy, kdy může při nedostatečném zabezpečení dojít ke značným finančním ztrátám. Tato práce se zabývá návrhem bezpečnostního řešení proti nejběžnějším útokům na VoIP systémy postavené na SIP protokolu. K realizaci tohoto řešení jsou především využity dva open source softwary Asterisk a Kamailio. Práce popisuje oba zmíněné programy a jejich možné využití v navrženém řešení. Tyto znalosti slouží k určení jejich úlohy ve finálním návrhu. Z velké části je práce věnovaná implementaci bezpečnostních funkcí a propojení obou softwarů do jednoho funkčního celku. Práce rovněž obsahuje detailní popis implementovaných funkcí a jejich analýzu. V neposlední řadě se práce zabývá výkonnostním testováním navrženého řešení a zhodnocením dosažených výsledků.

Klíčová slova

SIP, Asterisk, Kamailio, SBC, bezpečnost, výkonnostní testování

Abstract

Nowadays, implementing security policies is essential for all information systems connected to the Internet. VoIP systems are no exception. An insufficient level of security can lead to a significant financial loss. This thesis deals with the design of a security solution against the most common attacks on VoIP systems based on SIP protocol. To achieve this, two open source softwares are utilized, particularly Asterisk and Kamailio. First part of this thesis describes both Asterisk and Kamailio and their potential use. This knowledge is then used to determine their purpose in the final design. The main part is devoted to an implementation of these features and combining both softwares to work as one unit. In addition to the implementation of individual security mechanisms, the thesis also contains analysis of every feature and its advantages and possible shortcomings. The last part of this thesis is predominantly focused on benchmarking of the final solution.

Key words

SIP, Asterisk, Kamailio, SBC, security, performance testing

Obsah

Seznam použitých zkratek

Zkratka	Význam
AMD	Advanced Micro Devices
B2BUA	Back-to-Back User Agent
BRI	Basic Rate Interface
BSD	Berkeley Software Distribution – Berkeley Unix
CSV	Comma-Separated Values
DAHDI	Digium Asterisk Hardware Device Interface
DDOS	Distributed Denial of Service
DNS	Domain Name System
DOS	Denial of Service
DTLS	Datagram Transport Layer Security
FXO	Foregin eXchange Office
FXS	Foregin eXchange Subscriber
GPL	General Public License
HTTP	Hypertext Transfer Protocol
IDS	Intrusion Detection System
iLBC	Internet Low Bitrate Codec
IP	Internet Protocol
IPS	Intrusion Prevention System
ISDN	Integrated Services Digital Network
IVR	Interactive Voice Response
MiTM	Man in The Middle
NAPTR	Name Authority Pointer
NAT	Network Address Translation
PBX	Private Branch Exchange
PCM	Pulse Code Modulation
PKI	Public Key Infrastructure

PRI	Primary Rate Interface
PSTN	Public Switched Telephone network
RTP	Real Time Transport Protocol
SAR	System Activity Report
SBC	Session Border Controller
SDES	Session Description Protocol Security Descriptions
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SQL	Structured Query Language
SRTP	Secure Real Time Transport Protocol
SRV	Service Record
SSL	Secure Sockets Layer
TCP	Transport Control Protocol
TLS	Transport Layer Security
UA	User Agent
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
VM	Virtual Machine
VoIP	Voice over Internet Protocol
VPN	Virtual Private Network
ZRTP	Zimmermann Real Time Transport Protocol

Seznam obrázků

Seznam tabulek

1 Úvod

Voice over Internet Protocol je rychle rostoucí internetová služba díky nejen flexibilnímu a snadnému nasazení, ale také hlavně nižším nákladům ve srovnání s tradiční telefonní sítí. V současné době mnoho organizací upouští, nebo zcela upustily od tradičních telefonních systémů, a přechází k VoIP. Tento vývoj telefonie zvýšil zásadní potřebu bezpečnosti.

Za nejpopulárnější signalizační protokol ve VoIP lze označit SIP protokol. Hlavním důvodem je, že nabízí funkcionality s malou komplexností a vysokou škálovatelnosti. Jako většina protokolů, ani SIP není bez zranitelností. Toto je hlavní důvod, proč se v této práci zaměřím na realizaci bezpečnostního řešení k ochraně proti útokům na SIP protokol. K tomu především využiji dvou populárních open source softwarů, Asterisk a Kamailio.

Druhá kapitola nabízí popis bezpečnosti ve VoIP telefonii. Čtenář zde nalezne výčet nejčastějších typů útoků, se kterými se lze setkat při provozu VoIP serveru. Dále následuje popis základní definice SBC a výčet jeho funkcí s ohledem na bezpečnost VoIP.

Třetí kapitola je zaměřena na teoretický popis open source softwaru Asterisk. V úvodu kapitoly je rozebrána krátká historie vzniku softwaru a jeho původní účel fungování až po rozvoj do role, kterou v dnešní době může zastávat. V dalších dvou podkapitolách se zabývám architekturou Asterisku, a hlavně jeho možnosti zabezpečení.

Čtvrtá kapitola je věnována Kamailiu, jakožto druhému ze zmíněných softwarů využívaný v této práci. Podobně jako v kapitole o Asterisku, zde čtenář nalezne krátkou historii o vzniku a základní popis architektury. Další podkapitola je věnována modulům, a to především modulům, které poskytují různé bezpečnostní mechanismy k ochraně SIP protokolu.

Pátá kapitola je věnována návrhu a realizaci bezpečnostního řešení za pomocí obou zmíněných softwarů. V úvodu kapitoly dojde k identifikaci klíčových funkcí obou softwarů pro určení jejich úlohy ve finálním řešení. Dále bude rozebrána topologie testovací platformy a technické parametry jednotlivých serverů. To vše bude sloužit jako základ pro hlavní náplň této kapitoly, čímž bude realizace jednotlivých bezpečnostních funkcí, jejich analýza a testování.

Šestá kapitola se bude zabývat výkonnostním testováním navrženého řešení. Součástí kapitoly bude popis metodiky testování, použité nástroje pro sběr metrik, skripty a příkazy. Následně bude provedena analýza nasbíraných výsledků.

V závěru práce bude popsáno shrnutí úspěšně implementovaných funkcí a výsledků výkonnostního měření. Dále zde bude popis možného využití výsledného řešení, jeho flexibility a škálovatelnosti.

2 Bezpečnost VoIP telefonie

Útoků na VoIP infrastrukturu existuje celá řada. Je třeba si uvědomit, že VoIP systém se neskládá pouze z pobočkové ústředny a telefonů. Celá infrastruktura bude pravděpodobně zahrnovat různé části, od síťových zařízení, serverů, služeb až po jednotlivé protokoly. Všechny tyto části mohou potencionálně obsahovat zranitelnosti, které by nakonec mohly vést k ohrožení VoIP infrastruktury. Proto je nutné brát v úvahu bezpečnost celkové infrastruktury. Pro VoIP bude jedním z prvních kroků zabezpečení přenášené signalizace. V dnešní době jeden z nejpoužívanějších signalizačních protokolů pro realizaci VoIP služeb je SIP protokol. Právě na ochranu tohoto protokolu se tato práce zaměří.

2.1 Typy útoků

Cílem útoků na VoIP infrastrukturu je většinou snaha znepříjemnit uživatelům využívání dané služby nebo zneužít zranitelnosti systémů k různým podvodům pro finanční prospěch. Jak již bylo zmíněno, útoků existuje celá řada, proto budou následně popsány pouze významné hrozby a jejich dopad na funkčnost VoIP systému.

2.1.1 Odposlouchávání

Odposlouchávání existuje již od dob vzniku klasické telefonie a řadí se mezi jedny z nejoblíbenějších útoků. Ve VoIP můžeme, kromě odposlouchávání hlasových dat, zařadit také odposlouchávání signalizačního protokolu o právě probíhajícím hovoru.

Útok je možné realizovat v případě, že má útočník přístup k síťovému provozu. Například z Wi-Fi sítě, kde se používá volný prostor jako sdílené medium, které dělá zachytávání provozu velice jednoduché. Z tohoto zachyceného provozu může útočník získat signalizaci i hlasová nebo obrazová data hovoru.

Prevence proti těmto útokům je relativně jednoduchá. Nabízí se možnost v podobě šifrování jak signalizace SIP protokolu, tak i medií v podobě RTP paketů. Pro šifrování signalizace je určen TLS protokol a pro média protokol SRTP. Alternativou může být také přenesení celého provozu přes virtuální privátní síť VPN. Výhodu je, že jediný VPN tunel chrání SIP signalizaci i RTP média pro více relací. Nicméně sestavení VPN tunelu pro všechny komunikující strany z různých míst může být značně náročné, a ne vždy se jedná o vhodné řešení.[1]

2.1.2 Man in the middle

Man in the middle, zkraceně MiTM, je typem útoku, kdy se útočník nachází mezi dvěma komunikujícími stranami. Současně je schopen monitorovat, upravovat, vytvářet či mazat jakoukoli výměnu zpráv. Takový útok je v současné době obtížné realizovat. Ačkoliv ne zcela nemožné, má-li útočník kontrolu nad zařízením, přes které je směrován síťový provoz. K takovým útokům může docházet v případě nedostatečného či žádného zabezpečení, jako je absence šifrování či nesprávně nakonfigurované síťové prvky.

Příkladem takového útoku může být modifikace pole *Contact* v hlavičce SIP zprávy. Při registraci uživatele se toto pole využívá pro SIP URI daného uživatele. SIP server pak používá toto pole k vyhledávání daného uživatele, kde ho kontaktuje, obdrží-li požadavek na zpracování příchozího hovoru. Útočníkovi tedy stačí toto pole pozměnit, aby všechny příchozí požadavky na daného uživatele chodily právě na jeho zvolené zařízení.[2]

2.1.3 DoS útoky

DoS jsou útoky, jejichž hlavním cílem je omezit uživatelům využívat poskytované služby. Případně je omezit natolik, aby tyto služby byly zcela nedostupné. Většinou je tato akce dosažena zaplavováním severu nebo koncového zařízení obrovským množstvím nesmyslných požadavků, nebo zneužitím chyb v implementaci daného softwaru či protokolu.

Mezi záplavové útoky řadíme například generování velkého množství žádosti o registraci nebo TCP paketů s příznakem SYN. Tyto útoky jsou zaměřené především na vyčerpání zdrojů serveru v podobě procesorového času či množství využití operační paměti. U záplav registracemi stráví SIP registrar čas hledáním daného uživatele v databázi a vracením chybových odpovědí. Takové útoky lze dále zesílit na DDOS použitím takzvaných botnetů. Botnety jsou počítače, které již byly dříve infikované virem či trojským koněm, a útočník je využívá pro realizaci dalších útoků.

Při zneužití chyb v softwaru či protokolu není nutné pro odříznutí uživatelů od dané služby generovat velkého množství paketů. Zde postačí poslat několik správně cílených zpráv. Ačkoli úspěšnost této metody závisí na tom, zdali má útočník přístup k provozu v dané síti. Typickým druhem takového útoku je přesvědčení jedné ze stran o ukončení hovoru nebo odregistrování. V právě probíhajícím hovoru útočník využije zachycené hodnoty proměnných v záhlaví SIP zprávy pro vytvoření podvrhnuté zprávy *BYE* k ukončení hovoru. Případně k ukončení hovoru už během jeho vytváření může využít zprávy *CANCEL*. K odregistrování uživatele pak útočník zašle zprávu *REGISTER* s expirací nastavenou na hodnotu 0.[3]

2.1.4 Skenování

Skenování infrastruktury a účtů je nezbytným krokem každého útočníka. Díky této technice může získat dostatečné množství informací o dané síti, a tak se cíleně zaměřit na nejslabší části v této infrastruktuře. Skeny většinou bývají náhodné kontroly celého rozsahu IP adres a portů pro zjištění, jaké služby jsou na daném serveru dostupné. Jakmile na nějakou službu narazí, spouštějí se cílené útoky, které ze serveru mohou dostat nejrůznější informace. Běžný způsob skenování v SIP protokolu zahrnuje využití následujících zpráv:

- **OPTIONS** – posíláním těchto zpráv může útočník zjistit, zda daný server provozuje služby na SIP protokolu
- **REGISTER** – je pokus o registraci, na základě přijaté chybové odpovědi může být možné určit existující účty, u kterých se útočník pokusí o prolomení hesla
- **INVITE** – zjistí, zda server umožňuje spojit hovory bez nutnosti autentizace[4]

2.1.5 Další typy útoků

- **Replay útok** – útočník při přenosu dat zachytí pakety, které později zopakuje nebo pozmění přímo při průchodu
- **Phishing** – útočník se vydává za určitou důvěryhodnou entitu a snaží se od uživatelů získat důvěrné informace
- **SPIT** – je spam, tedy rozesílání nevyžádané reklamy prostřednictvím VoIP, při zaplnění hlasové schránky lze tento spam zařadit mezi DoS útok
- **Theft of service** – jedná se o stav, kdy má útočník přístup ke službám, které jsou za normálních okolností zpoplatněné nebo jinak limitované[2][3]

2.2 Session Border Controller

Session Border Controller, zkráceně SBC, je zařízení používané ve vybraných VoIP sítích k řízení kontroly nad signalizací, a také toku hlasových nebo vizuálních dat. Toto zařízení obvykle leží na okraji dvou sítí. Například dvou sítí poskytovatelů, kteří mezi sebou mají domluvený peering, mezi podnikovou sítí a sítí poskytovatelů, nebo mezi sítí rezidenčních uživatelů a sítí poskytovatelů. SBC často modifikují záhlaví SIP a těla zpráv, které běžné SIP proxy nemá důvod modifikovat. Dalším rozdílem oproti SIP proxy je možnost řízení toku médií. V důsledku tohoto faktu je SBC řazeno do definice B2BUA. SBC je také často označován jako SIP nebo VoIP firewall, jelikož častým důvodem jeho nasazení je právě implementace bezpečnostních politik a ochranných mechanismů proti nejrůznějším útokům.[5]

2.2.1 Maskování topologie

Maskování topologie spočívá v přepsaní nebo odstranění informací, ve kterých o sobě dávají vědět jednotlivá zařízení, v záhlaví SIP protokolu. Typicky se jedná o IP adresu nebo pole *user-agent*, kde například Asterisk uvádí používanou verzi. Tímto způsobem si útočník může dohledat, zdali tato verze neobsahuje nějaké zranitelnosti, a případně jich zneužít. Tato funkce je zejména užitečná pro poskytovatele, kteří nechtějí propagovat vnitřní infrastrukturu svým konkurentům nebo pro prevenci proti DoS útokům. V některých situacích se může jednat o požadavek ze strany zákazníka, který si nepřeje zveřejnit svou vlastní adresu.

Nejběžnějším způsobem skrytí topologie je odstranění všech předchozích polí *Via* a *Record-Route* v záhlaví SIP zprávy, nahrazení pole *Contact* nebo modifikace pole *Call-ID*. V polích, kde se používají místo doménových jmen IP adresy (pole *From* a *To*), které nelze odstranit, může adresy SBC nahradit vlastními.

Způsob skrytí topologie, jak je výše popsán, může v některých případech způsobovat problémy. Modifikaci záhlaví signalizační zprávy provádí SBC většinou bez uvědomění koncového uživatele. Může tak potencionálně upravit či odstranit informace související se zabezpečením, nebo s komunikaci s jinou aplikací. Dalším možným rizikem je ztráta stavu po odstranění polí ze záhlaví zprávy. Například z důvodu neočekáváného restartu, což může následně vést k neschopnosti opětovného navázání hovoru.[5]

2.2.2 NAT traversal

Pro dosažitelnost uživatelů umístěnými za NATem je SBC nucen modifikovat registrační informace daného uživatele. Typický uživatel za NATem do registrační zprávy uvádí privátní adresu. Příchozí hovory na tuto adresu by zajisté neuspěly, a to z důvodu, že adresa není směrovatelná ve veřejné Internetu. SBC tedy takovou privátní adresu nahradí za svoji vlastní veřejnou adresu, se kterou je následně uživatel registrován u SIP registraru. SBC také může provádět určitou formu periodické kontroly spojení tzv. keep-alive zprávy. Když SIP registrar obdrží žádost o registraci a reaguje kladnou odpovědí *200 OK*, může SBC v této odpovědi upravit expirační dobu registrace, aby přinutil uživatele obnovit registraci dříve, než by skutečně vypršela. Díky tomu se obnoví vazba na NAT před jejím vypršením. Alternativou může být využití SIP metody *OPTIONS*.

Podobně bude SBC zacházet i se zprávou INVITE v případech odchozích hovorů. V této zprávě od uživatele za NATem bude SBC modifikovat pole *Via* a *Contact,* kde nahradí privátní adresu uživatele za svoji vlastní.

V neposlední řadě je také nutné umožnit průchod RTP paketů přes NAT. Pro vyjednávání multimediálních vlastností spojení se používá SDP protokol, kde jeden z jeho atributu obsahuje IP adresu určující destinaci. Tuto adresu SBC opět nahradí za svoji vlastní. Takto se SBC stane prostředníkem pro RTP pakety a zaručí jejich správné doručení.[5]

2.2.3 Správa provozu médií

Správa provozu médií je funkce, která umožňuje SBC vynutit, aby tok médií přes něj procházel. Tato funkce je především užitečná pro poskytovatele VoIP služeb, kteří díky ní mohou vytvářet různé fakturační modely. Například videohovor může být jinak naceněný než běžný hlasový hovor apod. Další možnost je monitorování provozu ke shromažďování statistik k ujištění, že je dodržována dohodnuta kvalita služeb nebo pro audity a zákonné povinnosti.[5]

2.2.4 Šifrování

Samotný SIP protokol neobsahuje žádné zabezpečení proti Man in the middle útokům. Tentýž problém platí pro RTP protokol a jeho absenci ochrany proti odposlouchávání. Proto další funkci, kterou by SBC měl podporovat, je šifrování. Pro SIP signalizaci je ve většině případech využíván TLS protokol, kde se k ověření využívá princip distribuce veřejných klíčů z asymetrické kryptografie. Pro zabezpečení hlasových a obrazových dat se pak využívá SRTP, který RTP protokol rozšiřuje právě o bezpečnostní mechanismy.

2.2.5 Interoperabilita

Tato funkce umožňuje komunikaci mezi uživateli využívající různá zařízení s odlišnými technickými schopnostmi. Například SBC může umožnit spojení dvou uživatelů každý využívající jinou verzi IP protokolu nebo může zajistit konverzi různých kodeků.[5]

3 Asterisk

Asterisk je dnes nejrozšířenějším open source softwarem pro realizaci pobočkových telefonních ústředen (PBX) přenášející komunikaci jak za pomocí IP protokolu, tak i digitální ISDN nebo analogové telefonii. U zrodu Asterisku byl Mark Spencer, který se v roce 1999 snažil zaplnit díru na trhu provozem firmy LSS (Linux Support Services) provozující linku podpory pro operační systém Linux. Firma rychle rostla a zanedlouho potřebovala nový telefonní systém. S požadavkem se obrátil na místní prodejce telefonních ústředen, kteří mu však nabídli řešení značně převyšující jeho finanční rozpočet. Mark se tedy rozhodl vytvořit vlastní řešení. Po pár měsících usilovného vývoje vydal první funkční prototyp Asterisku, který zveřejnil a zpřístupnil pod licencí GPL. V roce 2001 změnil název firmy na Digium, která se dodnes i ve spolupráci s komunitou stará o vývoj Asterisku.[6]

Asterisk byl původně určený hlavně pro operační systém Linux, nicméně postupným vývojem se dostal i na další unixové systémy, jako OpenBSD, FreeBSD, NetBSD nebo Mac OS X. V populárnějších linuxových distribucích, jako Debian, Fedora, Red Hat nebo Ubuntu, je Asterisk dostupný v balíčkovém systému dané distribuce. Na ostatních distribucích je nutné jej zkompilovat. Dále existují i tzv. Asterisk distribuce, které se instalují jako běžný operační systém. Tyto distribuce nejsou nic jiného než minimálně upravená linuxová distribuce s předinstalovaným Asteriskem a jeho závislostmi. Příkladem takové distribuce je například 3CX založený na Debianu nebo FreePBX založený na CentOS.[7]

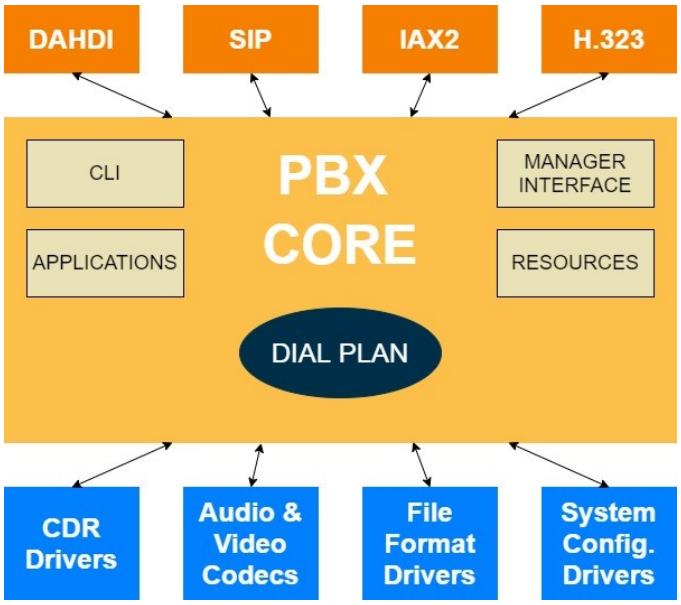
Od původního záměru, kde sloužil především jako pobočková telefonní ústředna, se Asterisk vyvinul do univerzálního nástroje s velkým množstvím komplexních funkcí. V současné době lze Asterisk provozovat v režimech:

- Pobočková ústředna (PBX)
- VoIP brána
- Voicemail služby
- Konferenční server
- Call centrum
- Interaktivní hlasový průvodce (IVR server)
- a mnoho dalších…[8]

3.1 Architektura

Asterisk se snaží poskytnout maximální flexibilitu pro různé telefonní technologie, proto je jeho architektura velmi odlišná od tradičních pobočkových ústředen. Veškerá vstupní a výstupní data z Asterisku prochází určitým typem kanálu. Kanál představuje logické komunikační rozhraní mezi jádrem Asterisku a různými zařízeními. Konkrétně se může jednat o telefon, pobočkovou ústřednu nebo jiný Asterisk server. Každý podporovaný protokol má v Asterisku svůj kanál v podobě modulu, jehož název se pojí s názvem daného protokolu, například *res_pjsip* pro protokol SIP nebo *chan_iax2* pro protokol IAX2. Směrem

k PSTN, kde se může jednat o ISDN BRI či PRI karty, FXS, FXO apod., využívá Asterisk kolekci open source ovládačů DAHDI a kanálový modul *chan_dahdi*.[7][9]



Obr. 3.1: Architektura Asterisku

3.1.1 **Dialplan**

Základ Asterisku tvoří dialplan. Jedná se o sadu instrukcí řídící jednotlivé kanály příchozích a odchozích hovorů a jejich směrování. Tvorba dialplanu probíhá v konfiguračním souboru *extensions.conf* formou skriptu se syntaxí specifickou pro Asterisk. Na rozdíl od tradičních telefonních systémů je plně přizpůsobitelný a nezávislý na použité technologie. Dialplan se skládá z kontextů, rozšíření, priorit a aplikací, které budou v následujících odstavcích popsány.[7]

V dialplanu jako první definujeme kontexty. Kontexty se zapisují do hranatých závorek a indikují začátek nové sekce dialplanu. Asterisk má dva předdefinované kontexty s názvem *general* a *globals*. V sekci *general*, jak už z názvu vyplývá, máme možnost definovat několik obecných možností týkajících se dialplanu, například ochranu vůči přepsání. V sekci *globals* definujeme konstanty a proměnné, které je možné později využít v dalších částech dialplanu. Další kontexty si administrátor definuje sám podle potřeby. Hlavní funkce kontextu je oddělení různých částí dialplanu, aby mezi sebou navzájem interagovaly. Administrátor si tak může nadefinovat zvlášť kontexty pro jednotlivé protokoly nebo rozlišit interní volání od externího.[7][9]

„Extensions“ znamená překladem rozšíření, volným překladem klapka, zařízení či telefonní linka. V tradičních pobočkových ústřednách se obvykle rozšíření vztahují k číselnému identifikátoru telefonů nebo hlasové schránky. Zatímco v dialplanu Asterisku tyto rozšíření představují určitou sadu instrukcí, kterou jsou postupně vykonávané definované akce. Syntaxe takových instrukcí může vypadat následovně:[7]

```
exten => název,priorita,aplikace()
exten => 200,1,Answer()
```

Asterisk nabízí poměrně velkou flexibilitu v pojmenovávání jednotlivých rozšíření. Název rozšíření může tedy být libovolnou kombinaci čísel a písmen. Ve výše zmíněné syntaxi je v názvu definovaný pouze jeden konkrétní název v podobě čísla *200*. To znamená, že by se daná akce vykonala pouze v případě vytočení tohoto čísla. Pro efektivitu zápisu velkého množství názvů Asterisk podporuje regulární výrazy. Zápis pak můžeme vypadat následovně:[7]

```
exten => _[0-9a-zA-Z].,1,Answer()
```

Pořadí vykonání jednotlivých instrukcí v dialplanu určují priority. Priority jsou číslovány postupně od čísla 1. Poté lze každé následující číslo zapsat jako písmeno *n*. Pokud Asterisk narazí na prioritu označenou *n*, vezme číslo předchozí priority, a následně k ní přičte 1. Toto značení umožňuje eliminovat problém přečíslovávání. V případě potřeby vložení nové instrukce mezi již očíslovanou sadu instrukcí. V rámci dalšího zjednodušení zápisu je možné nahradit *exten* novým operátorem *same* a vynechat název rozšíření za předpokladu, že je akce myšlena pro stejná rozšíření.[7]

```
exten => 200,1,Answer()
    same => n,Hangup()
```

Poslední části dialplanu jsou aplikace. Práce s aplikacemi je podobná jako při volání funkcí v některých programovacích jazycích. Každá aplikace v daném kontextu provádí určitou akci, například vytáčení, přehrání zvuku, čtení číselné volby zadávané uživatelem, vyhledávání v databázích, zavěšení hovoru apod. Stejně jako funkce některé aplikace přijímají specifické argumenty, bez kterých se neobejdou. Jiné zase žádné argumenty k provedení akce nepotřebují.[7]

3.1.2 **Moduly**

O modulech je nepřímé naznačení v úvodu této kapitoly. Kromě funkce jádra, moduly poskytují všechny ostatní funkce v Asterisku. Většina modulů je distribuována s instalací Asterisku. Ačkoliv další moduly mohou být dostupné od členů komunity nebo od firem, které poskytují vlastní komerční moduly. V konfiguračním souboru *modules.conf* je možné určit moduly, které se mají při startu Asterisku načíst, a také pořadí jejich načtení. Většina modulů má svůj vlastní konfigurační soubor, kde je možné nadefinovat vlastnosti specifické pro daný modul. Jednotlivé moduly jsou dle funkce řazeny do následujících typů:[7]

- **Aplikační moduly** – poskytují funkce různých akcí, které mohou být aplikované na hovor
- **Bridging moduly** – poskytují různé metody přemostění médií mezi účastníky hovoru
- **Call detail recording (CDR) moduly** – slouží k ukládání podrobných záznamů o hovoru
- **Channel event logging (CEL) moduly** – slouží pro sledování událostí souvisejících s kanálem
- **Kanálové moduly** – poskytují rozhraní do jádra Asterisku pro jednotlivé protokoly

- **Kodekové moduly** – umožňují překlad mezi různými kodeky hovoru
- **Moduly pro překlad formátů** – slouží k překladu formátu nahrávek
- **Dialplan moduly** – doplňují aplikační moduly a poskytují funkce jako manipulace se řetězci
- **PBX moduly** – jedná se o moduly jádra Asterisku, jejichž funkcí je například čtení konfiguračních souborů
- **Resource moduly** – zde patří všechny ostatní moduly, které se nehodí do výše zmíněných kategorií

3.2 Bezpečnost

Stejně jako pro většinu dnešních informačních systému, je bezpečnost pro Asterisk kritická. Obzvlášť, pokud je server dostupný z veřejné sítě. V následující kapitole budou popsány bezpečnostní mechanismy Asterisku, které nabízí pro své uživatelé.

3.2.1 B2BUA

B2BUA je speciálním typem UA, který je umístěný v signalizační cestě mezi dvěma účastníky. Jednotlivé požadavky účastníků jsou přeformulované, a následně odeslané jako nově vytvořené požadavky. Takto lze pomocí B2BUA implementovat maskovací službu, ve které mohou dvě strany mezi sebou komunikovat, aniž by se jedna ze stran dověděla o údajích druhé strany. Mimo přetvoření SIP zpráv dochází také k úpravě SDP tak, aby RTP pakety procházely přes B2BUA. Dalším možným využitím je tedy pro translaci kodeků nebo šifrování.[11]

3.2.2 Ochrana proti skenování účtů

Skenování účtů využívají faktu, že odpověď na registraci účtů přicházející ze serveru se liší v závislosti na tom, zda se jedná o existující účet či nikoliv. Pokud se požadavek na registraci obsahuje autentizační údaje existujícího účtu, server odpoví zprávou, kde vyžaduje autentizaci. V opačném případě server odmítne pokus o registraci. Toto chování je definované v samotném protokolu. A tím útočníkům umožňuje poměrně snadno zjistit existující účty. V Asterisku existuje mechanismus, který zajistí, aby odpověď na každou neúspěšnou žádost o registraci byla totožná. Takhle je pro útočníka skenování účtů bezcenné, jelikož se server tváří, jako by byl každý účet platný.[7]

3.2.3 Zabezpečení dialplanu

Dialplan je další z oblastí, kde je důležité vzít v úvahu zabezpečení. Jak už víme, dialplan lze rozdělit po částech díky kontextům. Důležité je do jednotlivých kontextů správně nadefinovat pouze volající, kteří by k dané službě měly mít přístup. Dalším potenciálním rizikem představuje nekorektní použití regulárních výrazů s předdefinovanými proměnnými. Příkladem je:[7]

```
exten => _X.,1,Dial(PJSIP/${EXTEN},30)
```

Tento regulární výraz odpovídá názvu všech zařízeních libovolné délky, které začínají číslem. Jelikož názvy nejsou omezené pouze na číselné znaky, potencionálnímu útočníkovi nic nebrání, aby vytočil následující řetězec "*500&PJSIP/itsp/771244555*". Tento řetězec by útočníkovi umožnil spojení do jeho zvolené destinace přes našeho poskytovatele internetových telefonních služeb. Jednoduchý způsob, jak se tomuto problému vyhnout je zápisem striktnějšího regulárního výrazu. Je-li to možné, povolíme pouze názvy do určité délky nebo pro specifickou množinu znaků. Vyhneme se tak rizikové konstrukci, která přijímá libovolné řetězce. Využití funkce FILTER() v dialplanu je další způsob, jak předejít vytáčení nechtěných rozšíření. Jak je z názvu funkce jasné, tato funkce umožňuje vyfiltrovat nechtěné znaky z názvu před tím, než se předají další instrukcí. Například následující konstrukce povolí vytočení pouze numerických názvu: [7][11]

```
exten => _X.,1,Set(SAFE_EXTEN=${FILTER(0-9,${EXTEN})})
exten => _X.,n,Dial(PJSIP/${SAFE_EXTEN})
```

3.2.4 Šifrování

Podpora šifrování signalizace pomocí TLS a multimediální data pomocí SRTP je samozřejmostí. Asterisk dokonce obsahuje pomocné skripty pro vytvoření vlastní certifikační autority, a následně i pro vygenerování podepsaného certifikátu potřebného pro zprovoznění TLS.

3.2.5 Log a Fail2Ban

Logy obsahují detailní záznam o činnosti programů a významně pomáhají při zpětné analýze příčiny problémů. Tyto záznamy se obvykle dělí to několika základních typů, například informační, varovné nebo chybové zprávy. Asterisk umožňuje poměrně flexibilní nastavení těchto zpráv. Každý typ zprávy je možné vypisovat zvlášť do vlastních souborů, konzole či do syslogu. Kromě základních typů zpráv Asterisk také poskytuje vypisování bezpečnostních zpráv. Obsah těchto zpráv je v dokumentaci podrobně definován ve snadně interpretovatelném formátu, který umožňuje jednoduchou integraci s externími nástroji. Jedním takovým nástrojem je Fail2ban.[7]

Fail2ban je linuxový nástroj schopný číst logy aplikace, a podle nich pak následně aktualizovat pravidla ve firewallu. Pokud z určité IP adresy chodí příliš mnoho požadavků, fail2ban je schopen vyhodnotit toto chování jako škodlivé. Následovně požadavky z dané IP adresy na nějaký čas blokuje. Výhodou je, že blokování proběhne už na síťové úrovni, čímž se zabrání zatěžování Asterisku agresivními boty. Další výhodou je možnost definovat skupinu IP adres, které by nikdy neměly být blokované. Fail2ban lze využít na většinu běžných útoků hrubou silou. Avšak nijak nezabrání distribuovaným útokům.[11]

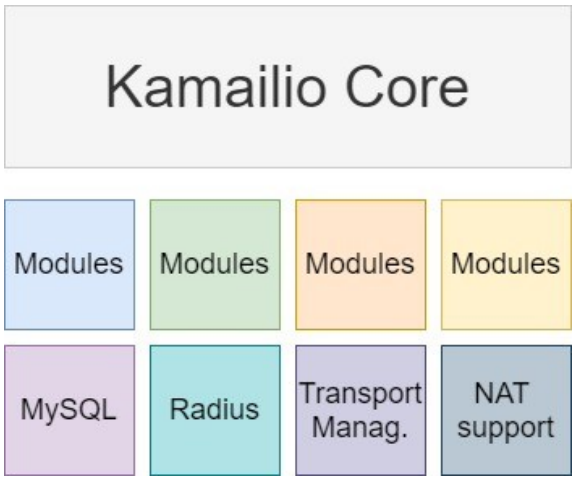
4 Kamailio

Kamailio je napsán jazyce C, na jehož vývoji se začalo pracovat už v roce 2001. V této době byl projekt vyvíjen berlínským výzkumným institutem pod název SER. Později se část vývojářů rozhodla přesunout do komerční společnosti *iptel.org*, a další část založila open-source projekt OpenSER. V roce 2008 byl kvůli problémům s obchodní značkou SER projekt OpenSER přejmenován na Kamailio, což v havajštině znamená mluvit či konverzovat. V listopadu téhož roku se vývojáři Kamailia a SER rozhodli založit projekt SIP router, který měl za úkol spojit vývojáře a uživatelskou komunitu obou projektů a sloučit zdrojové kódy pod jeden projekt. V roce 2010 byl projekt úspěšně dokončen, a následně byl vydán pod Kamailio ve verzi 3.0.0.[9][13]

Kamailio je především navržen pro zpracování SIP signalizace. Většinou zastává činnosti SIP proxy, SIP registraru nebo load-balancing serveru. Pomocí přídavných modulů je také možné pomocí Kamailia řídit tok hlasových a obrazových dat, ale funkcionalita je značně omezená. Proto se pro tyto funkcionality většinou využívá Asterisk. Ačkoliv v sobě neobsahuje tolik funkcí, jako výše zmíněný Asterisk, stále je hojně využíván pro svoji flexibilitu a výkonnost. Flexibilita spočívá v modulech, které je možné podle potřeby přidávat či odebírat bez nutnosti modifikace funkcí jádra. Výkon je dán samotnou definicí fungování SIP proxy, kde se server nestará o datový tok médií, což razantně snižuje nároky na hardware serveru. Dále je to způsob jeho fungování. Kamailio je po spuštění převeden do formy binárního souboru, který obsahuje veškerou funkcionalitu. Výsledkem je SIP server schopný obsloužit tisíce požadavků za sekundu.[14]

4.1 Architektura

Kamailio je, stejně jako Asterisk, stavěný na modulární architektuře. Zde opět figuruje jádro poskytující nízko úrovňové funkce a moduly, které obstárávají většinu dalších funkcí. Díky nímž se stává Kamailio velice výkonným SIP serverem.[9]



Obr. 4.1: Architektira Kamailia

4.1.1 Jádro

Jádro poskytuje funkce nižších vrstev, jako je například řízení paměti, rozhraní pro moduly nebo abstraktní vrstvu nad databázovými systémy. Několik funkcí jádra si následně popíšeme:

- **Správce paměti** – stará se o rozdělení dostupné paměti. Jakožto víceprocesorová aplikace architektura Kamailia využívá sdílené paměti pro přístup k proměnným, které využívají i ostatní procesy. U proměnných, které nemusí být sdílené pro ostatní procesy, je využito privátní paměti daného procesu.
- **SIP parser** – Kamailio obsahuje vlastní implementaci SIP parseru, také známý jako inkrementální (incremental) nebo líný (lazy) parser. Tento parser nezpracovává celé části SIP zprávy. Pro většinu záhlaví SIP identifikuje jeho název a tělo, ale nezkoumá obsah těla záhlaví.
- **Paser a interpet konfiguračního souboru** – k parsování konfiguračního souboru se využívají nástroje flex a bison. Jednotlivé příkazy jsou sestaveny do stromu akcí, které jsou prováděny proti každé SIP zprávě.
- **Řízení transportní vrstvy** – poskytuje podporu pro protokoly transportní vrstvy, jako je UDP, TCP nebo TLS. Dále poskytuje podporu pro DNS záznamy NAPTR a SRV dle požadavků uvedené v RFC3263.
- **Pseudo-proměnné –** jedná se o proměnné, které jsou určené výhradně pro čtení nikoliv pro zápis. Většina těchto proměnných odkazuje na informace uvnitř původní SIP zprávy, které se během exekuce konfiguračního souboru nemění.[13]

4.1.2 Moduly

Moduly jsou volitelné komponenty, které rozšiřují Kamailio o další funkce. Na rozdíl od Asterisku, kde má většina modulů vlastní konfigurační soubor, se veškerá konfigurace v Kamailiu provádí v jediném souboru. Moduly zde spíše představují knihovny s definovanými funkcemi, které se v konfiguračním souboru volají. Ve výsledku vypadá struktura tohoto souboru podobně jako program napsaný v jazyce s podobnou syntaxi jazyka C. Náročnost konfigurace je na vyšší úrovni než u Asterisku, proto se od uživatelů vyžaduje určitá znalost SIP protokolu.[13]

V poslední verzi Kamailia 5.3.x najdeme bezmála přes 200 modulů poskytující nejrůznější funkcionalitu, ať se jedná o účtování, autentizaci či bezpečnost. Popis všech modulů by byl velmi rozsáhlý, proto bude následující kapitola zaměřena pouze na moduly související s touto práci.

4.1.2.1 Htable modul

Tento modul přímo nenabízí bezpečnostní mechanismy, ale slouží jako doplněk pro ostatní moduly. Pomocí *htable* modulu je možné implementovat hashovací tabulky, které jsou uložené ve sdílené paměti. Přistupovat k nim lze pomocí pseudo-proměnných. Dalším typickým využitím je implementace cache systému, tj. pokud se již hledaná hodnota v hashovací tabulce nenachází, modul ji načte z databáze a následně ji do hashovací tabulky uloží. Tímto způsobem je následující přístup k této hodnotě velmi rychlý.

Tento modul také umožňuje jednotlivým hodnotám v hashovací tabulce nastavit expirační dobu, tudíž nedojde k zahlcení operační paměti.[15]

4.1.2.2 Permissions modul

Tento modul je ve své podstatě implementace IP a URI access listu. Množina povolených, resp. zakázaných, URI se zapisují do textového souboru ve formě regulárních výrazů s možností vyloučit určitou podmnožinu. Tyto pravidla *permissions* modul využívá k vyhodnocování oprávnění jednotlivých žádostí. Druhou možností je vyhodnocovat žádosti na základě důvěryhodných IP adres, které je možné nadefinovat do databáze.[16]

4.1.2.3 Pike modul

Tento modul udržuje záznamy všech (nebo vybraných) IP adres příchozích požadavků. Na základě těchto záznamů můžeme provádět blokaci. Za podmínky, že z dané IP adresy přichází velké množství požadavků v malém časovém intervalu. *Pike* modul přímo neprovádí blokaci, ale pouze zaznamenává nezvyklé vysoký provoz. Jakou akci provést si určuje sám správce za pomocí skriptování v konfiguračním souboru Kamailia.[17]

4.1.2.4 Pipelimit a ratelimit a moduly

Na rozdíl od *pike* modulu, který pracuje s IP adresami, moduly *pipelimit* a *ratelimit* omezují tok na základě příchozích SIP požadavků. Logika fungování těchto modulů je založena na algoritmech běžně používaných v síťových prvcích. Oba moduly podporují dva typy algoritmů pro omezování. Prvním typem jsou statické algoritmy:

- **Tail drop algoritmus** – jedná se o jednoduchý algoritmus, který zahazuje nově příchozí požadavky, jakmile jejich počet přesáhne nakonfigurovaný limit
- **Random early detection (RED) algoritmus** – tento algoritmus měří průměrnou zátěž, na základě, které dynamicky přizpůsobuje míru zahazování
- **Network algoritmus** – tento algoritmus spoléhá na informace poskytované síťovým rozhraním, míru zahazování vyhodnocuje na základě celkového množství příchozích bytů

Dalším typem je dynamický algoritmus:

- **Feedback algoritmus –** míru zahazování dynamicky vyhodnocuje na základě vytížení procesoru[18][19]

Modul *ratelimit* používá podobný princip jako firewall systémů BSD, kde je zajištěno řízení provozu přes tzv. pipes (linky) a queues (fronty). Jednotlivé metody SIP protokolu je možné spojit s frontou, která je následně přiřazená k lince s jedním z výše popsaných algoritmů. [19]

Modul *pipelimit* je odvozen s ratelimit modulu a přidává podporu pro definování limitů jednotlivých linek a dynamické názvy v databázi. Další změnou je vyřazení implementace front.[18]

4.1.2.5 Sanity check modul

Cílem tohoto modulu je implementovat několik základních kontrol formátování záhlaví příchozích požadavků. Tyto kontroly však nejsou pro funkčnost Kamailia vyžadovány. Na druhou stranu nedává velký smysl dále směrovat poškozené požadavky, pokud jsou dříve či později odmítnuté jiným SIP zařízením. Jelikož procházení všech polí v záhlaví zprávy může mít negativní vliv na výkon serveru, dává tento modul administrátorovi volnou ruku v tom, jaká pole by se měla kontrolovat.[20]

4.1.2.6 Secfilter modul

Sectfilter modul poskytuje úroveň zabezpečení pomocí blacklistu a whitelistu. Jednotlivé blokace, resp. povolení, je možné provádět na základě domén, IP adres, pole *user-agent*, uživatelů či zemí. Další užitečnou funkcí je prevence útoků SQL injekcí. Jednotlivé záznamy jsou uloženy do databáze, ze které se při startu Kamailia načtou do operační paměti serveru k zajištění rychlého přístupu. Záznamy je taky možné doplňovat či mazat během činnosti Kamailia.[21]

4.1.2.7 Topoh modul

Další modul pro zvýšení bezpečnosti VoIP sítě je *topoh* modul. Jeho cílem je skrytí topologie serverů přepsáním informací v záhlaví zprávy odhalující podrobnosti o topologií. Typicky se jedná o pole *Via*, *Contact*, *Record-route* nebo *Call-ID.* Z pohledu Kamailia se jedná o výpočetně nenáročný modul, který nemusí ukládat stavy jednotlivých zpráv a zachovává jejich transparentnost.[22]

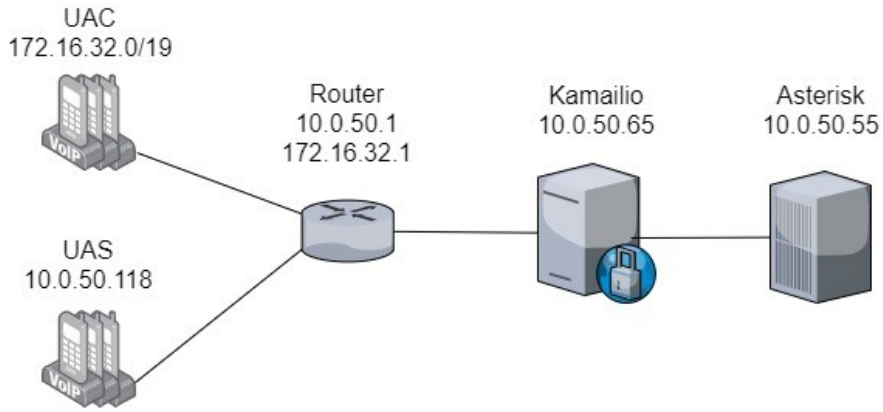
5 Návrh a realizace platformy

Z teoretické kapitoly jsme se dověděli definici SBC. Také jsme získali představu, jaké funkce by SBC měl plnit s ohledem na bezpečnost VoIP sítě. Tato část práce bude zaměřena na implementaci většiny vyjmenovaných bezpečnostních funkcí SBC za pomocí open source softwarů Asterisk a Kamailio.

Kamailio je v jádře především SIP proxy, jehož hlavní doménou je práce se SIP signalizací. Kdežto SBC je definován jako speciální typ B2BUA. Už z definice je tedy zřejmé, že samotné Kamailio nemůže zastávat funkci SBC. Zde přichází na řadu Asterisk, který naopak jako B2BUA funguje, a Kamailio o tuto část doplní. Systém se tedy bude skládat ze dvou hlavních prvků, a to Kamailio serveru a Asterisk serveru. Hlavní úlohou Kamailia bude řídit SIP signalizaci, registrace uživatelů a implementace většiny bezpečnostních funkcí. Veškeré požadavky budou tedy směrovat na tento server, kde se rozhodne, co s daným požadavkem provést, popřípadě kam jej přesměrovat. Za Kamailiem bude postavený Asterisk server, jehož hlavní roli bude zajistit spojení příchozích hovorů a zpracovávat tok mediálních dat. To také zahrnuje šifrování RTP paketů a translaci mezi různými kodeky. Z pohledu uživatelů se bude signalizace a RTP tok tvářit, jako by pocházely z jednoho serveru.

Jak již bylo v úvodu kapitoly zmíněno, topologie platformy se bude skládat ze serverů Asterisku a Kamailia. Přičemž budou oba tyto servery virtualizované na jediném fyzickém stroji za pomocí virtualizačního nástroje Oracle VM VirtualBox. Logická struktura celého návrhu je zobrazena na obrázku 5.1. Technické parametry virtuálních serverů budou vypadat následovně:

- 2 jádra z AMD ryzen 7 2700
- 4 GB DDR4 RAM
- 1 Gbit/s Ethernet
- Debian 10 buster 64-bit



Obr. 5.1: Logická struktura platformy

5.1 Instalace Asterisku a Kamailia

Oba programy lze pomocí příkazů *apt* nebo *aptitude* nainstalovat z repozitářů operačního systému Debian, kde však vždy nebývá nejstabilnější verze. Proto budeme oba programy kompilovat ze zdrojových kódů. Pro Asterisk byla zvolena nejnovější LTS (long-term support) verze v době psaní této práce, konkrétně se jedná o verzi 16.6.2. Pro Kamailio pak byla zvolena verze 5.3.2. Podrobný postup instalace obou softwarů může čtenář nalézt v příloze této práce.

Po instalaci jsou konfigurační soubory Asterisku umístěné v */etc/asterisk*. Jako odrazový můstek lze využít ukázkové konfigurační soubory v instalační složce Asterisku pod *configs/samples* nebo *configs/basic-pbx*. A nakonec v */user/local/etc/kamailio/* najdeme konfigurační soubory Kamailia.

5.1.1 Asterisk - pjsip.conf

V tomto souboru se definují jednotlivé sekce pro uživatele, jako jsou například autentizační údaje, kontaktní údaje či transportní protokol. Konfigurace bude poměrně jednoduchá. Jediným úkolem Asterisku bude přijmout požadavek z Kamailia, sestavit hovor a poslat jej zpět. Lokalizaci jednotlivých uživatelů pak provádí Kamailio, proto zde postačí nadefinovat pouze jeden koncový bod, na který bude Asterisk směrovat veškeré příchozí a odchozí zprávy.

```
;===TRANSPORT===
[udp-transport]
type = transport
protocol = udp
bind = 10.0.50.55

[tcp-transport]
type = transport
protocol = tcp
bind = 10.0.50.55

;===KAMAILIO===
[kamailio]
type = endpoint
context = kamailio
disallow = all
allow = alaw
allow = g726
direct_media = no
from_domain = 10.0.50.65
```

```
[kamailio]
type = identify
match = 10.0.50.65
endpoint = kamailio
```

V sekci TRANSPORT definujeme IP adresu a transportní protokol, na které bude Asterisk přijímat příchozí požadavky. Není-li specifikován port, Asterisk bude poslouchat na výchozím portu pro SIP protokol. V další sekci už definujeme koncový bod s názvem *„kamailio*“, pro který jsme povolili kodeky G.711 A-law a G726. Dále parametrem *direct_media* říkáme, aby Asterisk mezi uživatele nevytvářel přímé spojení. Pomocí parametru *from_domain* můžeme změnit doménu v poli *From*. Nakonec definujeme sekci *identify*, která zajistí, aby Asterisk po Kamailiu nevyžadoval autentizaci.

5.1.2 Asterisk - extensions.conf

Tento soubor slouží k definování pravidel, která směrují jednotlivé hovory do specifické destinace (viz. 2.1.1). Možnosti pro tvorbu dialplanu jsou poměrně rozsáhlé. Rozebírat jej však do detailu není součástí této práce. Pro účely této práce postačí níže uvedené řádky. Těmito řádky zajistíme správné sestavení hovoru a následné předání na Kamailio, aby jej přesměroval na koncového uživatele.

```
[kamailio]
exten => _[0-9a-zA-Z].,1,
Dial(PJSIP/kamailio/sip:${EXTEN}@${SIPDOMAIN}, 10)
```

5.1.3 Konfigurace Kamailia a propojení s Asteriskem

Konfigurace Kamailia probíhá především pomocí dvou souborů, a to *kamctlrc* a *kamailio.cfg*. V prvním jmenovaném konfiguračním souboru se definují proměnné pro ovládací nástroje *kamctl* a *kamdbctl*. Pro naše účely bude dostačující nastavit a odkomentovat řádky pro nastavení domény a připojení k MySQL databázi.

```
SIP_DOMAIN = 10.0.50.65
DBENGINE = MYSQL
DBHOST = localhost
DBRWUSER = "kamailio"
DBRWPW = "StrongPassword"
```

V tomto konkrétním případě *SIP_DOMAIN* představuje IP adresu serveru, na kterém je Kamailio realizován. Další řádky nám určují typ SQL serveru, jeho adresu a přístupové údaje pro uživatele, který bude moct z této databáze číst nebo do ní zapisovat.

Hlavní konfigurace funkcionalit se provádí v druhém z jmenovaných konfiguračních souborů, *kamailio.cfg*. Tento soubor se dělí do sekcí s globálními parametry, parametry pro moduly a sekci pro směrování požadavků. Kontrolu nad spouštěním jednotlivých bloků kódu řídí Kamailio pomocí direktiv

převzatých z jazyka C. Tento způsob řízení zajišťuje nižší využití paměti a rychlejší exekuci příkazů. Mezi další výhody můžeme zařadit snadnější způsob, jak povolit či zakázat funkce nebo přepnutí z bloku, kde nebyly splněné podmínky, do druhého. Ve výchozím nastavení konfigurační soubor Kamailia obsahuje mnoho užitečných direktiv, které je možné využít. Dle potřeby může uživatel dodefinovat další direktivy.

Na začátku nastavíme IP adresu a porty, na kterých bude Kamailio poslouchat. Dále vypneme automatické vyhledávání lokálních aliasů. Jelikož Kamailio ve výchozím nastavení provádí reverzní DNS vyhledávání ke zjištění názvu hostitelů pro jednotlivé IP adresy a snaží si je uložit jako aliasy.

```
auto_aliases = no
listen = udp:10.0.50.65:5060
listen = tcp:10.0.50.65:5060
listen = tls:10.0.50.65:5061
```

Dále propojíme Kamailio s MySQL databází. Naštěstí základní konfigurace obsahuje skoro všechny potřebné parametry. Proto bude stačit dodefinovat direktivu a doplnit správné přístupové údaje, které byly nastavené v *kamctlrc* souboru.

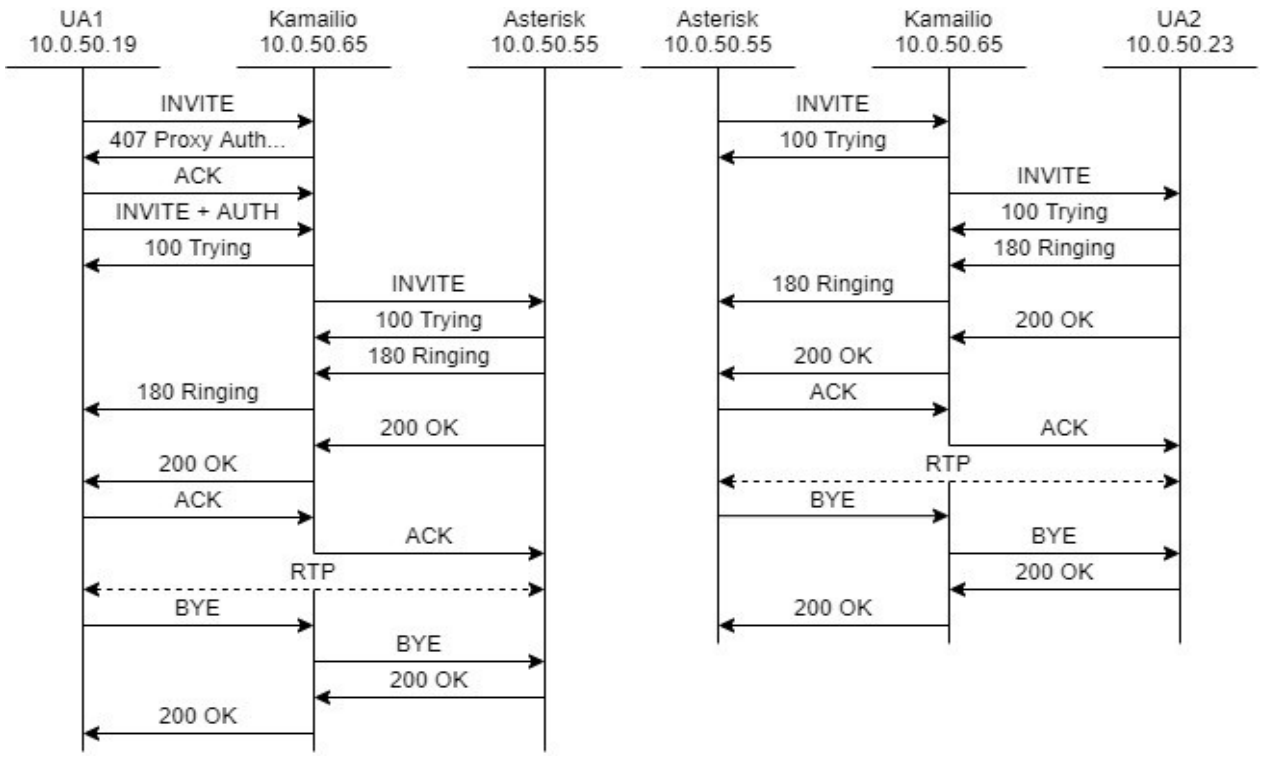
```
######## GLOBAL PARAMETERS #######
...
#!define WITH_MYSQL
...
#!ifdef WITH_MYSQL
#!define DBURL"mysql://kamailio:StrongPassword@localhost/kamailio
"
#!endif
```

Nyní můžeme začít nastavování propojení mezi Kamailiem a Asteriskem. Zde bude nastavení o něco náročnější, protože všechny vlastnosti budeme muset ručně dodefinovat. Nejprve v sekci s globálními parametry nadefinujeme novou direktivu a IP adresu s portem, na které bude Asterisk server poslouchat pro příchozí požadavky.

```
#!define WITH_ASTERISK
...
#!ifdef WITH_ASTERISK
asterisk.bindip = "10.0.50.55" desc "Asterisk IP address"
asterisk.bindport = "5060" desc "Asterisk Port"
#!endif
```

V sekci pro směrování nastavíme dvě směrovací cesty. První cesta *FROMASTERISK* určuje směr z Asterisku a druhá *TOASTERISK* směr k Asterisku. Dále v bloku *AUTH* říkáme Kamailiu, aby po Asterisku nevyžadoval autentizaci. Nakonec v bloku *LOCATION* zajišťujeme přeposílání požadavků směrem k Asterisku. Toto přeposílání bylo nastaveno pouze pro signalizační zprávy typu *INVITE*, tedy požadavky na hovor, jelikož tuto funkcionalitu poskytuje Asterisk. Výsledný call flow je zobrazen na obrázku 5.2.

```
route[AUTH] {
#!ifdef WITH_ASTERISK
    if($si==$sel(cfg_get.asterisk.bindip)
       && $sp==$sel(cfg_get.asterisk.bindport))
    {
        return;
    }
#!endif
...
}
route[LOCATION] {
#!ifdef WITH_ASTERISK
    if(is_method("INVITE") && (!route(FROMASTERISK)))
    {
        route(TOASTERISK);
        exit;
    }
#!endif
...
}
#!ifdef WITH_ASTERISK
route[FROMASTERISK] {
    if($si==$sel(cfg_get.asterisk.bindip)
       && $sp==$sel(cfg_get.asterisk.bindport))
    {
        return;
    }
    return -1;
}
route[TOASTERISK] {
    $du="sip:" + $fU +"@" $sel(cfg_get.asterisk.bindip) + ":"
        + $sel(cfg_get.asterisk.bindport);
    route(RELAY);
    exit;
}
#!endif
```



Obr. 5.2: Ukázka signalizace hovoru

5.2 Maskování topologie

V ideálním případě maskování topologie je z veřejné sítě viditelný pouze jeden prvek, přes který směruje veškerá signalizace a tok médií. Podíváme-li se zpětně na obrázek 5.2, můžeme vidět, že v případě signalizace máme splněno. Jednotliví uživatelé komunikují s Kamailiem, který funguje jako proxy a stará se o směrování do určené destinace. U toku RTP paketů můžeme vidět, že tok médií vynechává server s Kamailiem, a je směrován přímo na Asterisk, který funguje v režimu B2BUA. Z pohledu skrývání topologie je využití B2BUA výhodné. Z principu jeho fungování pomáhá skrývat uživatelé. Z tohoto důvodu se nemusíme zabývat skrýváním IP adres jednotlivých uživatelů. Nezbytné bude pouze nakonfigurovat skrývání adresy Asterisk serveru.

5.2.1 RTPProxy

Následně nastavíme Kamailio tak, aby hrál roli prostředníka i pro RTP pakety. Pro dosažení tohoto cíle využijeme nástroj RTPProxy. Jeho instalaci lze provést pomocí příkazu *apt*, jelikož se nachází v repozitářích debianu.

Nastavení nástroje následně provedeme v souboru */etc/default/rtpproxy*, kde do řádku *EXTRA_OPTS* přidáme parametry, se kterými se nástroj bude spouštět.

```
EXTRA_OPTS="-l 10.0.50.65 -m 8000 -M 30000 -s udp:127.0.0.1:7722
-d INFO -u rtpproxy"
```

- **l** – nastaví IP adresu, kde RTPProxy poslouchá
- **m** – je dolní limit intervalu pro porty paketů RTP protokolu
- **M** – je horní limit intervalu pro porty paketů RTP protokolu
- **s** – je ovládací socket nástroje a slouží například k nastavení vazby s Kamailiem
- **d** – tento parametr nastavuje úroveň výřečnosti v logu
- **u** – určuje uživatele, pod kterým se nástroj spouští

Pro umožnění spolupráce Kamailia s RTPProxy, musíme povolit stejnojmenný modul, který zajistí propojení obou softwarů přes nastavený ovládací socket. Načtení modulu do Kamailia a nastavení připojení provedeme dle níže uvedených řádků v konfiguračním souboru *kamailio.cfg*.

```
loadmodule "rtpproxy.so"
...
# ----- rtpproxy params -----
modparam("rtpproxy", "rtpproxy_sock",
"unix:/var/run/rtpproxy.sock")
```

K popisu vlastností mediální relace slouží protokol SDP, který ve svém těle nese potřebné informace pro navázání spojení. Z tohoto hlediska nás budou zajímat především dvě pole, a to pole s atributem „*o*“ a „*c*“. Obě tyto pole obsahují IP adresu, na kterou je tok RTP paketů směrován. Níže je zobrazen příklad SIP zprávy obsahující SDP zprávu zkrácenou na relevantní položky.

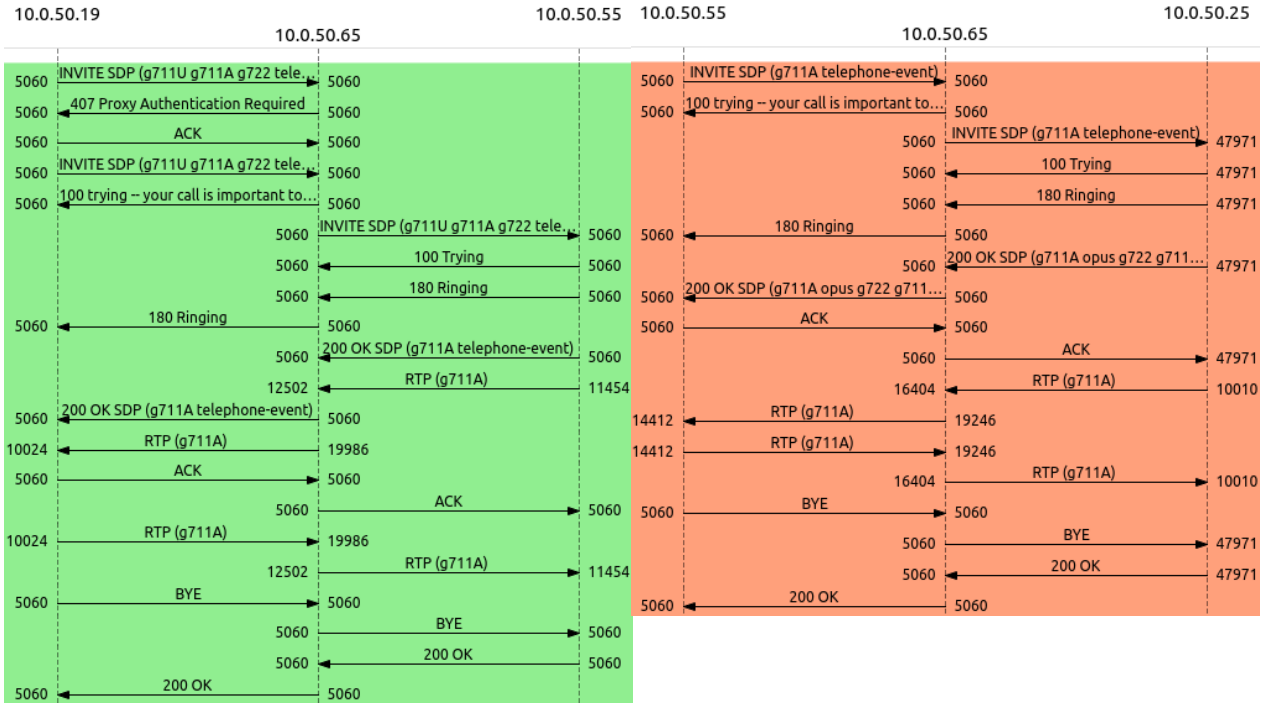
```
INVITE sip:alice@10.0.50.65 SIP/2.0
Via: SIP/2.0/UDP 10.0.50.19:5060;branch=z9hG4bK1998763059;rport
...
o=bob 8000 8000 IN IP4 10.0.50.19
s=SIP Call
c=IN IP4 10.0.50.19
...
```

K přepsaní této adresy na adresu Kamailia v našem případě 10.0.50.65, zajistíme pomocí funkce *rtpproxy_manage("co")* dostupné z načteného modulu *rtpproxy*. Tento příkaz je pak nutné vložit do funkcí spravující příchozí žádosti, pozitivní a negativní odpovědi na ně.

```
route[RELAY] {
    rtpproxy_manage("co");
...
}
onreply_route[MANAGE_REPLY] {
    rtpproxy_manage("co");
...
}
```

```
failure_route[MANAGE_FAILURE] {
    rtpproxy_manage("co");
...
}
```

K otestování funkčnosti výše implementované funkce si vygenerujeme a odchytíme hovor mezi dvěma účastníky. Hovor bude probíhat mezi Alici (10.0.50.25) a Bobem (10.0.50.19). Na levé straně obrázku 5.3 si můžeme všimnout první část hovoru pocházejícího od Boba. Na pravé straně pokračuje druhá část směřující z Asterisku na Alici. Ze signalizace lze vidět, že oba účastnící směřují jak signalizaci, tak i RTP tok směrem na Kamailio, který se stará o další směrování. Výsledná signalizace je tedy shodná se signalizací zobrazena na obrázku 5.2. Jediným rozdílem je RTP tok, který prochází přes RTPProxy na serveru s Kamailiem.



Obr. 5.3: Testovací hovor

5.2.2 Topoh

Pro skrytí zbývajících informací o Asterisku v záhlaví signalizačních zpráv využijeme modulu *topoh*. Načtení modulu se provádí stejným způsobem jako u ostatních modulů, a to příkazem *loadmodule*. V parametrech modulu nastavíme klíč, pomocí kterého budou jednotlivé zprávy zamaskované anebo odmaskované. Dále nastavíme IP adresu použitou pro zamaskování původní adresy a napojení na *sanity* modul. Nemá smysl totiž provádět maskování u špatně naformátovaných zpráv. Vhodné je také zmínit, že maskovací adresa by neměla být adresa potencionálně využívána jiným zařízením.

```
loadmodule "topoh.so"
...
# ----- topoh params -----
modparam("topoh", "mask_key", "ceecaiHi4Equi7k")
modparam("topoh", "mask_ip", "127.0.0.10")
modparam("topoh", "sanity_checks", 1)
```

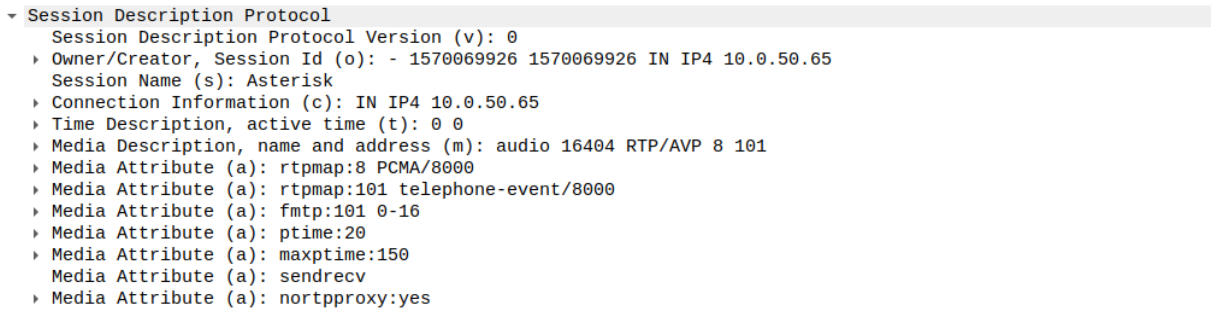
Takto nastavený *topoh* modul bude provádět maskování pro všechny odchozí zprávy, což zahrnuje i zprávy směrem k Asterisku. Tímto směrem však není nutné provádět maskování. Proto udělíme následující funkcí výjimku pro IP adresu 10.0.50.65, tedy adresu Asterisku.

```
event_route[topoh:msg-outgoing] {
    if($sndto(ip) == "10.0.50.55") {
        drop;
    }
}
```

Velkou výhodou tohoto modulu je způsob, jakým provádí maskování topologie oproti variantě popsané v RFC 5853. Místo mazání jednotlivých polí, jsou tato pole ponechávána a pouze se v nich přepisují informace odhalující topologii vnitřní sítě. Pro směrování jednotlivých zpráv nemá přepisování vliv. Odmaskování se provádí před spuštěním konfiguračního skriptu. Maskování se naopak provádí až po dokončení. V praxi to znamená, že jsou jednotlivé zprávy zpracovány v podobě, jak jsou nastaveny volajícím nebo volaným.

Na obrázku 5.4 je zobrazena zpráva *INVITE* z Asterisku směrující k Alici. Tato zpráva obsahuje pozměněna pole *Via* a *Contact* k zamaskování IP adresy Asterisku.

```
▾ Session Initiation Protocol (INVITE)
  ▸ Request-Line: INVITE sip:alice@10.0.50.25:47971;rinstance=d720aeef6b5c5a5e;transport=UDP SIP/2.0
  ▾ Message Header
    ▸ Record-Route: <sip:10.0.50.65;lr;ftag=0bb15628-0742-4b47-83c9-802498e8912e;nat=yes>
    ▸ Via: SIP/2.0/UDP 10.0.50.65;branch=z9hG4bKe868.7bcb3fd9fa74be46076e86379f3a8a87.0
    ▸ Via: SIP/2.0/UDP 127.0.0.10;branch=z9hG4bKsr-WsPNZ8OfMUh2jyanMgafMUKtMUKtDgQtMeImYzqPIb2X7J23ig9mZ-afDgafDgWp
    ▸ From: <sip:bob@10.0.50.65>;tag=0bb15628-0742-4b47-83c9-802498e8912e
    ▸ To: <sip:alice@10.0.50.65>
    ▸ Contact: <sip:127.0.0.10;line=sr-lbPmYJdoLSyXLGP1Ne9mZ-afDgafDgWkDgabMedoLGPol8s5MUKmZ-WmZ-Wt6-WmD-BAMN**>
      Call-ID: 9b80d30a-caf0-4da8-aa40-459edcaa86e0
    ▸ CSeq: 5555 INVITE
      Allow: OPTIONS, REGISTER, SUBSCRIBE, NOTIFY, PUBLISH, INVITE, ACK, BYE, CANCEL, UPDATE, PRACK, MESSAGE, REFER
      Supported: 100rel, timer, replaces, norefersub
      Session-Expires: 1800
      Min-SE: 90
      Max-Forwards: 69
      User-Agent: Some generic B2BUA
      Content-Type: application/sdp
      Content-Length:   251
  ▾ Message Body
```



Obr. 5.4: Ukázka maskování topologie

5.3 Ochrana vůči DoS a MiTM

5.3.1 Htable a Pike

Nejčastějším typem DoS útoku bývá generování velkého množství provozu k zatížení cílového serveru. Právě na obranu proti takovým útokům se v této kapitole zaměříme. Pro realizaci ochrany použijeme modul *pike* a *htable*. Oba tyto moduly jsou již ve výchozí konfiguraci nadefinované pod direktivou *WITH_ANTIFLOOD*. K načtení modulů tedy stačí direktivu povolit.

Z hlediska *pike* modulu jsou významné parametry *sample_time_unit* a *reqs_density_per_unit*. První z těchto dvou parametrů určuje interval v sekundách použitý pro analýzu počtu příchozích požadavků. Druhý parametr navazuje na první a určuje maximální počet povolených požadavků za tento interval. V praxi bude Kamailio blokovat IP adresu, pokud z ní přichází více než 20 požadavků v intervalu dvou vteřin. Poslední parametr *remove_latency* souvisí s tím, jak tento modul pracuje. Každá blokovaná IP adresa je rozdělena po jednom bytu do binárního stromu a uchovávána v paměti po dobu určeném právě tímto parametrem. Díky tomu je možné analyzovat IP adresy ze stejného rozsahu mnohem rychleji.

```
# ----- pike params -----
modparam("pike", "sampling_time_unit", 2)
modparam("pike", "reqs_density_per_unit", 20)
modparam("pike", "remove_latency", 60)
```

V kapitole o modulech Kamailia bylo řečeno, že *pike* neprovádí přímou blokaci. Tuto funkcionalitu zajistíme pomocí modulu *htable* a částečně skriptováním. V parametrech pro modul *htable* nadefinujeme hashovací tabulku s názvem *ipban*. Velikostně tato tabulka pojme 4096 = 2^12 záznamů, kde bude mít každý z nich nastavenou automatickou expiraci na 10 min.

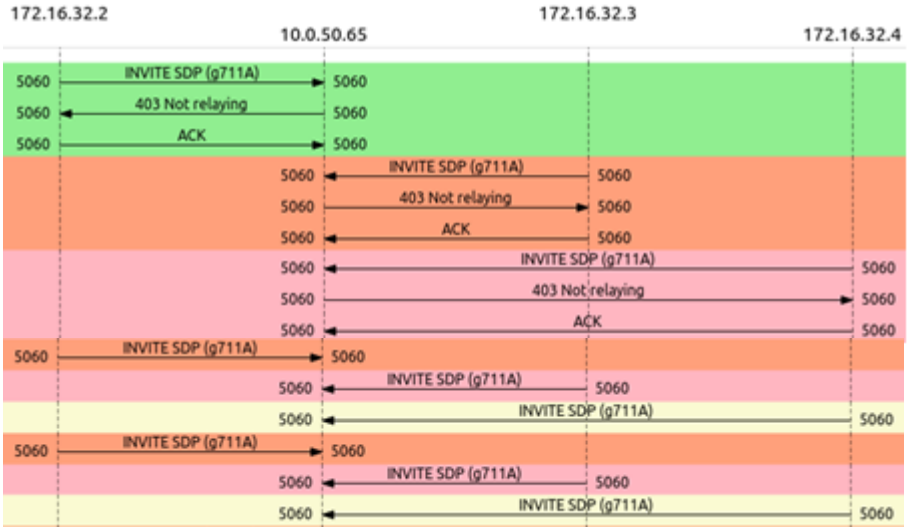
```
# ----- htable params -----
modparam("htable", "htable", "ipban=>size=12;autoexpire=600;")
```

Níže jsou úpravy ve skriptu zajišťující blokaci IP adresy v momentě, kdy překročí limit *pike* modulu. Tato kontrola je zaručena pomocí funkce *pike_check_req()*. Jakmile tento limit překročí některá

adresa, zapíše se do hashovací tabulky záznam. Tento záznam zůstává v paměti po dobu nastavenou v expirací modulu *htable*. Následně se vytvoří záznam v logu a skript se ukončí. Další požadavky z této adresy se kontrolují proti záznamům v hashovací tabulce. Je-li IP adresa v tabulce nalezena, opět se vytvoří záznam v logu a skript se ukončí.

```
route[REQINIT] {
if(src_ip!=myself) {
      if(!allow_source_address("10") && $sht(ipban=>$si)!=$null) {
            xlog("L_NOTICE",
            "Request $rm from blocked user $fu (IP:$si:$sp)\n");
            exit;
      }
      if (!allow_source_address("10") && !pike_check_req()) {
            $sht(ipban=>$si) = 1;
            xlog("L_WARN",
            "Blocking $rm from $fu (IP:$si:$sp)\n");
            exit;
      }
       }
}
```

K otestování funkčnosti ochrany využijeme nástroje SIPp. Scénář testu bude jednoduchý. Virtuální stroj simulující UAC bude posílat na Kamailio zprávy INVITE s různými IP adresami z rozsahu 172.16.32.2-11 překračující limit *pike* modulu. Na obrázku 5.5 je uvedena ukázka signalizace ze zachyceného provozu. Pro přehlednost, ukázka obsahuje pouze část z blokovaných IP adres a moment, kdy dochází k překročení limitu, a následnému blokování. Jakmile je tento limit překročen, Kamailio už na další zprávy z dané IP adresy neodpovídá.



Obr. 5.5: Ukázka DoS útoku

Tato ochrana je efektivní, do doby než útočník provoz deleguje mezi větší množství zařízení. Každé z těchto zařízení má vlastní adresu a generuje malé množství požadavků. Sumarizaci všech těchto požadavků je provoz dostačující ke zahlcení serveru. Tento typ útoků je klasifikovaný jako DDoS útok a *pike* modulem je těžko detekovatelný. Limit se totiž váže na IP adresu, a v případě jeho značného snížení může docházet k blokování legitimních uživatelů. Nutno je také podotknout, že ochrana proti DDoS útokům by se měla za správných okolností implementovat před cílovým serverem vhodnými IDS/IPS systémy. V momentě, kdy je server zcela zahlcený požadavky, je velmi obtížné provádět jakoukoliv obranu.

5.3.2 GeoIP

Čím si můžeme do určité míry pomoct je *GeoIP* modul. Tento modul umožňuje provádět dotazy v reálném čase proti MaxMind GeoIP2. Jedná se o databázi s IP adresami přiřazené k určitým geografickým místům, které lze získat zdarma nebo komerčně na stránkách[1] firmy MaxMind. Neočekáváme-li od uživatelů, že by se připojovali z jiné země než z České republiky, je možné IP adresy z ostatních zemí preventivně, díky tomuto modulu, zablokovat. Po zavedení modulu nastavíme cestu k souboru databáze GeoIP2. A následně nadefinujeme do skriptu podmínku, která bude provádět kontrolu.

```
loadmodule "geoip2"
...
modparam("geoip2", "path",
"/usr/local/etc/kamailio/geoip/GeoLite2-City.mmdb")
...
route[REQINIT] {
if(geoip2_match("$si", "src")) {
      if(($gip2(src=>cc))!="CZ") {
            xlog("L_INFO",
            "Request $rm originating from: $gip2(src=>cc)\n");
            send_reply("403", "Forbidden");
            exit;
         }
    }
...
}
```

[1] https://dev.maxmind.com/geoip/geoip2/geolite2/

5.3.3 TLS a SRTP

U dalšího typu DoS útoku může útočník ukončit probíhající hovor podvrženou zprávou *BYE* nebo *CANCEL*. Pro ochranu vůči takovému útoku slouží šifrování signalizace pomocí protokolu TLS. Mimo tuto ochranu, šifrování také poskytuje ochranu proti Man in the middle útokům. Proti odposlouchávání pak slouží SRTP protokol, který šifruje přenášená média.

Šifrovanou komunikaci je nutné zajistit především mezi klienty a Kamailiem. Nutnost šifrování mezi Asteriskem a Kamailiem se odvíjí podle topologie dané sítě. Pokud komunikace mezi servery probíhá přes nedůvěryhodnou síť, jako je internet nebo lokální síť, kterou nemáme plně pod kontrolou, je vhodné šifrování mezi servery zajistit. V našem konkrétním případě není šifrování mezi těmito servery nezbytné, protože jejich komunikace probíhá v rámci jediného fyzického stroje. Z tohoto důvodu bude šifrování povoleno pouze mezi klienty a Kamailiem. K povolení šifrované komunikace pomocí TLS protokolu musíme nejprve Kamailiu zajistit certifikát a privátní klíč. V případě, že máme k dispozici doménu, je možné využít certifikační autority Let‘s Encrypt, která vydává certifikáty zcela zdarma. V rámci lokální nebo firemní sítě je možné využít OpenSSL k vytvoření vlastní certifikační autority, klíčů či certifikátů. Celý proces vytváření certifikační autority může čtenář nalézt v příloze této práce. Následně uvedeme pouze žádost o certifikát, a jak ji nechat podepsat certifikační autoritou.

```
$ openssl req -config openssl.cnf -new -out kamailio.csr -keyout \
kamailio.key -days 1098 -newkey rsa:2048
$ openssl ca -config openssl.cnf -in kamailio.csr -out kamailio.crt
```

První z uvedených příkazů generuje žádost o certifikát s platností na tři roky. Při vytváření požadavku vyplňujeme identifikační údaje, kde je pole *CommonName* nejdůležitější. Zde se vyplňuje doména či IP adresa serveru. V případě, že se název serveru neshoduje s tímto polem je certifikát neplatný. Druhým příkazem už pouze necháváme žádost podepsat certifikační autoritou. Certifikát s klíčem je možné ponechat ve formátu *.crt* resp. *.key*, nebo jej můžeme vložit do totožného souboru s formátem *.pem*.

Šifrování v Kamailiu explicitně povolíme řádkem *enable_tls*. Následně načteme modul pro práci s TLS protokolem a cestu k certifikátu a klíči. Poslední parametr určuje, jaké verze TLS protokolu je možné využít k šifrovaní komunikace. Vhodné je zajistit využívání TLS verze 1.2 nebo vyšší. Nižší verze jsou nachylné na různé druhy útoků a jejich podpora se ve světě pomalu ukončuje.

```
enable_tls = yes
loadmodule "tls.so"

# ----- tls params -----
modparam("tls", "private_key",
"/usr/local/etc/kamailio/kamailio.key")
modparam("tls", "certificate",
"/usr/local/etc/kamailio/kamailio.crt")
modparam("tls", "tls_method", "TLSv1.2+")
```

Zabezpečení médií pomocí SRTP protokolu budeme provádět na Asterisk serveru. Podporu pro SRTP v Asterisku zajišťuje modul *res_srtp.so*. Přidáním následujícího řádku do souboru *modules.conf* zajistíme načtení tohoto modulu při startu Asterisku.

```
load = res_srtp.so
```

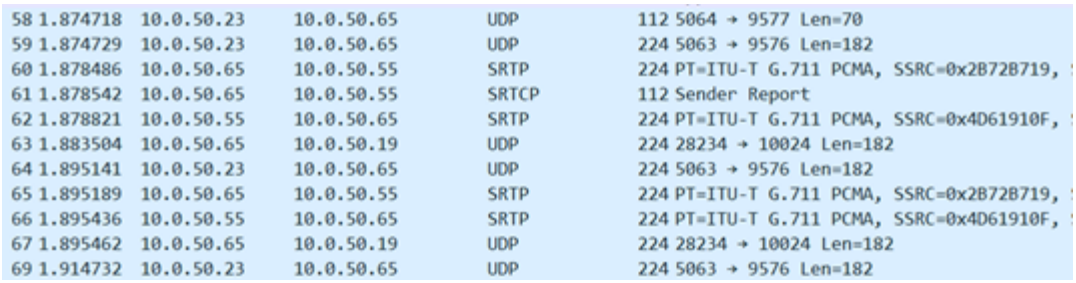
Následující nastavení probíhá v konfiguračním souboru *pjsip.conf*. Do sekce, kde máme nadefinovaný koncový bod pro Kamailio, přidáme *media_encryption* s jednou z následujících možností:

SDES – je jednoduchý mechanismus pro vyjednání šifrovacích klíčů pro SRTP. Při použití tohoto mechanismu je naprosto nezbytné, aby signalizace byla šifrovaná pomocí TLS protokolu. Klíče určené k šifrování médií jsou totiž přenášené SDP protokolem v SIP zprávě v čistě textové podobě.

DTLS – je další možnost pro vyjednání šifrovacích klíčů. Klíče se však nepřenáší přes SDP protokol v textově podobě, ale je zde využit princip PKI. Jedná se o princip infrastruktury, která důvěřuje určité certifikační autoritě a její vydaným certifikátům. V tomto případě je přenášená pouze veřejná část šifrovacího klíče.

Pro naše účely využijeme mechanismu SDES. Díky výše zprovozněnému šifrování SIP signalizace bude výměna šifrovacích klíčů dostatečně zabezpečena. K otestování opět vytvoříme hovor mezi Alici a Bobem. Na obrázku 5.6 je zobrazena ukázka ze zachyceného provozu, kde lze vidět kombinaci šifrované a nešifrované komunikace. Mezi šifrovanou komunikaci patří zprávy mezi Kamailiem (10.0.50.65) a oběma účastníky (Alice – 10.0.50.23, Bob – 10.0.50.19). Dále k této komunikaci patří UDP pakety, které jsou ve skutečnosti SRTP pakety. Kvůli šifrované signalizace Wireshark tyto pakety neumí dát do kontextu, proto je nesprávně zobrazuje. Nešifrovaná signalizace pak probíhá mezi Asteriskem (10.0.50.55) a Kamailiem, kde už naopak vidíme správně zobrazené SRTP pakety.

15	0.429983	10.0.50.19	10.0.50.65	TCP	15…	43878 → 5061 [ACK] Seq=1688 Ack=471 Wi
17	0.430660	10.0.50.19	10.0.50.65	TLSv1.2	172	Application Data
18	0.431367	10.0.50.65	10.0.50.19	TLSv1.2	402	Application Data
20	0.431667	10.0.50.65	10.0.50.55	SIP/SDP	181	Request: INVITE sip:alice@10.0.50.65:5
21	0.432236	10.0.50.55	10.0.50.65	SIP	608	Status: 100 Trying |
23	0.433112	10.0.50.55	10.0.50.65	SIP/SDP	10…	Request: INVITE sip:alice@10.0.50.65 |
24	0.433440	10.0.50.65	10.0.50.55	SIP	403	Status: 100 trying -- your call is imp
25	0.434134	10.0.50.65	10.0.50.23	TLSv1.2	16…	Application Data
27	0.435310	10.0.50.23	10.0.50.65	TLSv1.2	938	Application Data
29	0.451312	10.0.50.23	10.0.50.65	TLSv1.2	11…	Application Data
31	0.451584	10.0.50.65	10.0.50.55	SIP	896	Status: 180 Ringing |
32	0.452163	10.0.50.55	10.0.50.65	SIP	793	Status: 180 Ringing |
33	0.452433	10.0.50.65	10.0.50.19	TLSv1.2	794	Application Data
37	0.647038	10.0.50.140	239.255.255.250	UDP	77	35939 → 15600 Len=35
51	1.870766	10.0.50.23	10.0.50.65	TLSv1.2	14…	Application Data
53	1.871231	10.0.50.65	10.0.50.55	SIP/SDP	12…	Status: 200 OK |
54	1.872022	10.0.50.55	10.0.50.65	SIP	709	Request: ACK sip:alice@10.0.50.23:5062
55	1.872128	10.0.50.55	10.0.50.65	SIP/SDP	12…	Status: 200 OK |
56	1.872376	10.0.50.65	10.0.50.23	TLSv1.2	733	Application Data
57	1.873128	10.0.50.65	10.0.50.19	TLSv1.2	12…	Application Data



58	1.874718	10.0.50.23	10.0.50.65	UDP	112 5064 → 9577 Len=70
59	1.874729	10.0.50.23	10.0.50.65	UDP	224 5063 → 9576 Len=182
60	1.878486	10.0.50.65	10.0.50.55	SRTP	224 PT=ITU-T G.711 PCMA, SSRC=0x2B72B719,
61	1.878542	10.0.50.65	10.0.50.55	SRTCP	112 Sender Report
62	1.878821	10.0.50.55	10.0.50.65	SRTP	224 PT=ITU-T G.711 PCMA, SSRC=0x4D61910F,
63	1.883504	10.0.50.65	10.0.50.19	UDP	224 28234 → 10024 Len=182
64	1.895141	10.0.50.23	10.0.50.65	UDP	224 5063 → 9576 Len=182
65	1.895189	10.0.50.65	10.0.50.55	SRTP	224 PT=ITU-T G.711 PCMA, SSRC=0x2B72B719,
66	1.895436	10.0.50.55	10.0.50.65	SRTP	224 PT=ITU-T G.711 PCMA, SSRC=0x4D61910F,
67	1.895462	10.0.50.65	10.0.50.19	UDP	224 28234 → 10024 Len=182
69	1.914732	10.0.50.23	10.0.50.65	UDP	224 5063 → 9576 Len=182

Obr. 5.6: Ukázka šifrování provozu

5.4 Kontrola formátování SIP zpráv

Kontrola formátovaní zpráv je již ve výchozí konfiguraci Kamailia implementovaná. Tato kontrola je prováděna pomocí funkce *sanity_check* ze *sanity* modulu. Jednotlivé kontroly, která tato funkce může provádět, jsou rozdělené do mocnin čísla 2. Funkce pak přijímá jako argument součet těchto čísel. Volitelně funkce také přijímá druhý celočíselný argument určující, jaké URI budou zkontrolované. Tento argument má význam pouze u provádění kontroly s číslem 1024. K dispozici jsou následující kontroly:

- (1) – kontroluje verzi SIP protokolu v příchozím požadavku
- (2) – kontroluje RURI
- (4) – kontroluje, zda požadavek obsahuje povinná pole
- (8) – nevyužíváno
- (16) – nevyužíváno
- (32) – kontroluje, zda je metoda v poli *Cseq* shodná s metodou zprávy
- (64) – kontroluje, zda číslo v poli *Cseq* je kladné celé číslo
- (128) – kontroluje, zda se velikost těla zprávy odpovídá hodnotě v poli *Content-length,*
- (256) – kontroluje, zda číslo v poli *expires* je kladné celé číslo
- (512) – zkontroluje, zda zpráva obsahuje všechny položky z pole *proxy-require,*
- (1024) – kontroluje, zda všechny URI jsou pro Kamailio čitelná
  - (1) – *Request* URI
  - (2) – *From* URI
  - (4) – *To* URI
- (2048) – kontroluje všechny výskyty autentizačních údajů ve zprávě
- (4096) – kontroluje, zda záhlaví obsahuje duplicity pole *To* a *From*
- (8192) – kontroluje pole *Authorization*
- (16384) – kontroluje správnost pole *Via*

Celá podmínka s funkcí vypadá následovně:

```
if(!sanity_check("21991", "7")) {
    xlog("Malformed SIP response from $si:$sp\n");
    exit;
}
```

Kontrola jednotlivých polí se provádí pro příchozí zprávy a jejich odpovědi. Některé kontroly jsou pro odpovědi vynechány, jelikož z pohledu těchto zpráv nemají význam. Následující tabulka ukazuje rozklad součtu na jednotlivé kontroly pro oba směry:

	Mocniny čísla 2														
Součet hodnot	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
Příchozí zprávy - 21991	1	0	1	0	1	0	1	1	1	1	0	0	1	1	1
Odpovědi - 21700	1	0	1	0	1	0	0	1	1	0	0	0	1	0	0

Tab. 1 Kontroly pro příchozí zprávy a jejich odpovědi

5.4.1 SQL injekce

S SQL injekcí se nejčastěji můžeme setkat v rámci protokolu HTTP, protože server má tendenci používat informace k získání, vložení, aktualizace nebo odstranění dat z databáze. Použití databáze je pro SIP protokol podobné, a pro jednotlivé uživatelé se ukládají nezbytné informace k zajištění různých služeb. Nejběžnější využívanou tabulkou v Kamailiu je *Subscriber* tabulka. Tato tabulka ukládá informace o uživateli, jako je například uživatelské jméno, heslo nebo doména. Díky tomu je perfektním cílem pro SQL injekci. Potencionální útočník by mohl do tabulky podvrhnout dalšího uživatele, který by umožnil přístup k určitým službám, anebo naopak z tabulky některé informace smazat.

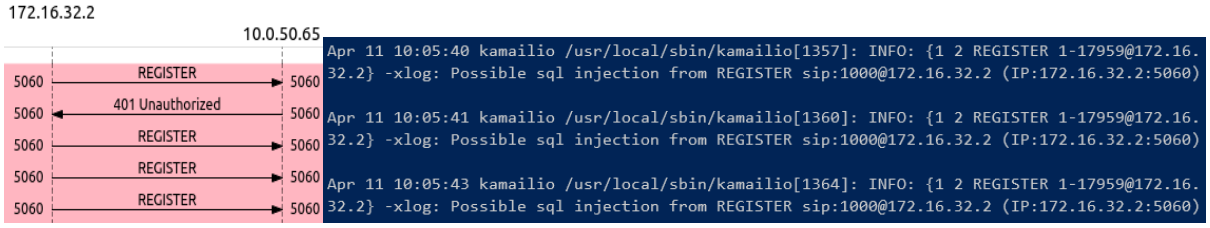
Pro ochranu proti těmto útokům lze využít modul *secfilter*, konkrétně jeho funkce *secf_check_sqli_all*. Tato funkce poskytuje základní ochranu vůči SQL injekci. Při bližším prozkoumání zdrojového kódu této funkce lze zjistit, že již za administrátora provádí veškeré nezbytné kroky. V první řadě dochází k vyhledávání nelegálních znaků v poli *User-agent*, *From*, *To* a *Contact* záhlaví SIP zprávy. U nálezu ilegálních znaků je zapsán záznam do logu, a následně je zpráva zahozena. Využití je tedy triviální a funkci stačí ve skriptu umístit do místa, kde se provádí kontrola příchozích zpráv. Jediným nedostatkem je absence kontroly pro pole *Authorization*, které také může potencionálně nést data pro SQL injekci. Podmínku pro kontrolu tohoto pole tedy budeme muset ručně dopsat.

```
route[REQINIT] {
...
secf_check_sqli_all();
if($au =~ "(\=)|(\-\-
)|(')|(\#)|(\%27)|(\%24)|(\%60)" && $au != $null) {
        xlog("L_INFO",
        "Possible sql injection from $rm $fu (IP:$si:$sp)");
        exit;
}
...
}
```

Pro otestování budeme posílat zprávu *REGISTER* pomocí nástroje SIPp s upraveným polem *Authorization*. Toto pole bude obsahovat SQL kód pro změnu uživatelského jména „bob“ na „hacker“. Níže lze vidět, jak bude toto pole vypadat.

```
Authorization: Digest username="1000;UPDATE subscriber SET
username='hacker' where username='bob'--",
realm="172.16.32.2"ri="sip:10.0.50.65:5060",
nonce="XpF6LF6RefDrqObG/C7RbL/DHulKarbEX5/dDYA=",
response="1171f398209f4de7c8a245a98072f034",alrithm=MD5
```

Následující obrázek 5.7 zachycuje chování Kamailia. Ze signalizace lze vidět, že Kamailio na tyto zprávy neodpovídá, a před jejich zahozením provede zápis záznamu do logu.



Obr. 5.7: Ukázka blokování SQL injekce

5.5 Uživatelské účty – skenování a hádání hesel

K zamezení útočníkovi získat seznam existujících účtů slouží jednoduchý mechanismus, který obsahuje jak Asterisk, tak i Kamailio. Celý princip spočívá v tom, že SIP registrar by měl odpovídat stejnou chybovou zprávu bez ohledu na to, zda bylo nesprávné uživatelské jméno či heslo. Samozřejmě je možné, že útočník jméno uživatele získal jiným způsobem, a následně se pokusí metodou brute force (útok hrubou silou) o prolomení hesla k tomuto účtu. Přičemž existují dva typy chování útočníků. Jeden bude SIP registrar zaplavovat velkým množstvím registračních zpráv, kde každá z nich bude zkoušet jiné heslo. Toto excesivní zasílání zpráv lehce zachytí již implementovaný *pike* modul. Druhý typ útočníka je trpělivější a posílá požadavky v menším množství, aby neaktivoval limit *pike* modulu. Proti takovému útočníkovi zavedeme pro registrace základní bezpečnostní politiku. Při překročení počtu neúspěšných registrací bude tento uživatel po nějakou dobu zablokován.

Pro správnou implementaci u této bezpečnostní politiky je třeba si uvědomit, jak probíhá registrace v SIP protokolu. Typická registrace uživatele obsahuje dvě zprávy *REGISTER*. První zpráva slouží k iniciaci registrace a neobsahuje autentizační údaje. SIP registrar na tuto zprávu odpovídá zprávou *401 Unauthorized* s výzvou obsahující informace, které musí být použité k šifrování hesla uživatele. Po téhle výzvě uživatel posílá druhou zprávu *REGISTER* s polem *Authorization*. Na základě správnosti údajů, v tomto poli je uživatel buď registrován, nebo odmítnut.

K implementaci opět využijeme hashovací tabulku, do které budeme ukládat počet neúspěšných pokusů a čas poslední neúspěšné registrace. Pro kontrolu autentizačních údajů slouží následující podmínka:

```
if(!auth_check("$fd", "subscriber", "1")) {
      auth_challenge("$fd", "0");
       exit;
}
```

Do této podmínky zapadnou registrační zprávy se špatnými autentizačními údaji, ale i počáteční registrační zprávy, kvůli absenci pole *Authorization*. Proto tuto podmínku musíme upravit tak, aby se počitadlo neúspěšných pokusů zvyšovalo pouze u registrací obsahující pole *Authorization* s nesprávnými autentizačními údaji. V případě, že se uživatel správně zaregistruje během deseti povolených pokusů, počitadlo se resetuje, resp. nezačne počítat. Výsledná úprava podmínky bude vypadat následovně:

```
route[AUTH] {
...
 if(!auth_check("$fd", "subscriber", "1")) {
      if($sht(userban=>$au::attemps) == $null) {
           $sht(userban=>$au::attemps) = 0;
       }
       if(is_present_hf("Authorization")) {
           $sht(userban=>$au::attemps) =
               $sht(userban=>$au::attemps) + 1;
           xlog("L_INFO", "Failed auth - incrementing count for
               $fu to: $sht(userban=>$au::attemps)");
       }
       if($sht(userban=>$au::attemps) >= 10) {
           xlog("L_WARN", "Too many failed auth in a row from
               $fu (IP:$si:$sp)\n");
       }
       $sht(userban=>$au::last_auth) = $Ts;
       auth_challenge("$fd", "0");
       exit;
 }
...
}
```

V konečném kroku je nutné zajistit kontrolu, zda daný uživatel nepřekročil limit neúspěšných registrací. V takovém případě je tento uživatel zablokován po dobu 15 minut.

```
route[REQINIT] {
...
if($sht(userban=>$au::attemps) >= 10) {
  $var(exp) = $Ts - 900;
      if($sht(userban=>$au::last_auth) > $var(exp)) {
          xlog("L_NOTICE",
          "USERBAN - Request $rm from blocked user
          $fu (IP:$si:$sp)\n");
          sl_send_reply("403", "Try later");
```

```
            exit;
        }
        else {
            $sht(userban=>$au::attemps) = 0;
        }
}
...
}
```

5.5.1 Registrování uživatelů a blacklist

K výše uvedené implementované politice zavedeme další politiku, která povoluje určité funkce pouze registrovaným účastníkům. V opačném případě jim bude Kamailio vracet chybovou zprávu v podobě *403 Forbidden.* Dále budeme pomocí modulu *secfilter* v databázi udržovat černou listinu. Tato listina se může vztahovat na doménu, IP adresu, uživatele nebo zemi. Pro ukázku implementujeme černou listinu uživatelů. Jednotlivé uživatele je možné na listinu přidat pomocí ovládacího nástroje *kamcmd*. První z níže uvedených podmínek tedy kontroluje, zdali je daný uživatel zaregistrovaný. Druhá podmínka provádí kontrolu uživatelů proti zavedené černé listině. Pokud se uživatel na listině nachází, dojde k zahození jeho požadavku.

```
route[REQINIT] {
...
if(src_ip!=myself) {
    if(is_method("INVITE|OPTIONS|SUBSCRIBE")) {
        if(!registered("location", "$fu")) {
            sl_send_reply("403", "Forbidden");
            exit;
        }
    }
}
secf_check_ua();
if($? == -2) {
    xlog("L_ALERT",
    "Request $rm from blacklisted user $ua (IP:$si:$sp)\n");
    exit;
}
...
}
```

6 SIPp – test výkonu

SIPp je bezplatný open source testovací nástroj pro generování SIP signalizace. Tento nástroj je především optimalizován pro výkonnostní testování SIP serverů. Avšak jej lze využít i pro vygenerování jednoduchého hovoru, kde si může uživatel ověřit, zda hovor neselže. SIPp je dostupný pro všechny unixové operační systémy. Windows je také podporován, ale zde SIPp nedosahuje optimálního výkonu.[14]

Pro generování testovacího provozu využívá SIPp předem nadefinované scénáře v XML souboru. Ve výchozí instalaci již má nástroj několik základních scénářů k základnímu otestování funkčnosti jak tohoto programu, tak i SIP serveru. Pro komplexnější testování je možné vytvořit vlastní scénáře. Scénáře se zapisují pomocí jazyku XML a ukládají do souboru se stejnojmennou příponou. Jednotlivé příkazy jsou definované pomocí XML párových značek tzv. *tags*. Například odesílání zpráv provádí příkaz *<send></send>*. Zpráva k odeslání se zapisuje mezi tento příkaz, který je následně vložen do pole CDATA. Formát SIP zprávy klasicky podléhá doporučení RFC 3261. Značnou výhodou je možnost do SIP zpráv nadefinovat, místo statických hodnot, dynamické proměnné. Pro využití velkého množství proměnných, například různé IP adresy nebo přihlašovací údaje, SIPp umožňuje načítaní těchto proměnných z externího souboru. K tomu slouží speciální proměnná *fieldX*, kde *X* je číslo označující sloupec v *CSV* souboru.[23]

Nástroj je původně zamýšlen pouze pro generování SIP signalizace, proto je generování hlasových dat implementováno poměrně v omezené míře. Pro simulaci odesílaní těchto dat může uživatel vygenerovat *pcap* soubor, který bude obsahovat zachycená hlasová data ve formě RTP paketů zakódovaných pomocí určitého kodeků. Tento soubor lze snadno získat pomocí nástroje Wireshark. Od verze 3.4 nástroj umožňuje generování hlasových dat ze zvukového souboru (např. soubor WAV) zakódovaného kodekem PCMA, PCMU, G722, iLBC nebo G729.[23]

6.1 Příprava serverů

Pro generování zátěže je nutné spustit minimálně jednu instanci nástroje SIPp v roli UAC a další v roli UAS. Obě tyto instance je možné spouštět na jednom výkonnějším počítači. Nicméně pro jednodušší přerozdělení systémových prostředků bude pro každou instanci dedikovaný vlastní virtuální stroj. Jelikož nástroj SIPp nepodporuje paralelizaci, budou parametry pro tyto dva servery skromnější než u Kamailia a Asterisku. Na těchto serverech si vystačíme s těmito parametry:

- 1 jádro z AMD ryzen 7 2700
- 2 GB DDR4 RAM
- 1 Gbit/s Ethernet
- OS Debian 10 buster 64-bit

Dále z důvodu simulace produkčního prostředí, a také kvůli limitu *pike* modulu, bude mít server s UAC přidělené adresy z rozsahu 172.16.32.0/19.

6.1.1 Souborový limit

V Linuxu je každá entita považována za soubor. To znamená, že vše v systému od procesů, souborů, adresářů až po sockety je vyjádřeno abstraktním indikátorem, nazývaný souborový deskriptor. Z bezpečnostních důvodu je pro jednotlivé uživatelé a aplikace nastaven limit počtu současně vytvořených deskriptorů. V systému Debian je limit nastaven na hodnotu 1024 a pro účely testování je nedostačující. Proto jej na všech serverech navýšíme. Limit je možné změnit v souboru */etc/security/limits.conf*, kde doplníme následující řádky:

```
* hard nofile 150000
* soft nofile 150000
```

Další limit, který musíme navýšit, se týká nástroje *RTPProxy*. Tento nástroj má ve výchozím nastavení limit na 1024 deskriptorů. Tento počet je možné navýšit pomocí přepínače „-L“, ale k tomu nástroj vyžaduje zvýšená oprávnění. Proto kromě tohoto přepínače změníme uživatele, pod kterým se *RTPProxy* běžně spouští na uživatele *root*.

```
EXTRA_OPTS="-l 10.0.50.65 -m 8000 -M 30000 -s udp:127.0.0.1:7722
-d INFO -u root -L 10000"
```

6.2 SIPp scénáře

6.2.1 UAS

Dříve než začneme psát scénář, připravíme *CSV* soubor, odkud SIPp bude číst potřebné informace. CSV soubor bude mít na první řádku definovaný parametr, který říká SIPp, v jakém pořadí má jednotlivé řádky číst. Tento parametr může nabývat jedné z těchto tří hodnot – SEQUENTIAL, *RANDOM* nebo *USER*. U možnosti *SEQUENTIAL* čte SIPp soubor *CSV* řádek po řádku. Dojde-li na konec souboru, začne jej číst od začátku. Možnost *RANDOM* říká SIPp, aby soubor četl v náhodném pořadí. Poslední možnost *USER* je uživatelem definovaný vzorec pro pořadí čtení. Pro naše účely využijeme první možnost *SEQUENTIAL*.

Nakonec zbývá do souboru doplnit informace, které bude SIPp používat při generování hovorů. Mezi tyto informace bude patřit číslo volajícího, jméno a heslo pro autentizaci. Finální *CSV* soubor bude mít následující podobu:

```
SEQUENTIAL
10000;[authentication username=10000 password=test];
10001;[authentication username=10001 password=test];
...
```

Tvorbu scénáře začínáme řádkem s definicí kódování. Druhý řádek *DOCTYPE* je nepovinný a slouží k určení *DTD* soubor pro kontrolu syntaxe. Dále následuje blok *scenario* s pojmenováním scénáře. Do tohoto bloku vkládáme jednotlivé příkazy v podobě párových značek *XML* jazyka.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">
<scenario name="uas-register">
```

Níže je uveden příklad, který bude registrovat nového uživatele na SIP registar. Přihlašovací údaje si tento scénář bude načítat z předem vytvořeného *CSV* souboru. Odesílání provádíme příkazem *send*. Uvnitř tohoto příkazu si může uživatel nadefinovat další pomocné parametry. Například *start_rtd="register"* spouští interní časovač s názvem register nebo parametr *retrans="500"*, který říká SIPp, aby zprávu znovu přeposlal, nedostane-li odpověď do 500 ms.

```
<send retrans="500" start_rtd="register">
<![CDATA[

REGISTER sip:[local_ip] SIP/2.0
Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
Max-Forwards: 70
To: <sip:[field0]@[local_ip]>
From: <sip:[field0]@[local_ip]>;tag=[call_number]
Call-ID: [call_id]
CSeq: 1 REGISTER
Contact: <sip:[field0]@[local_ip]:[local_port]>
Expires: 3600
Content-Length: 0
User-Agent: Sipp
]]>
</send>
```

Mezi tímto blokem se nachází pole CDATA, kde je SIP zpráva již formátovaná dle doporučení RFC 3261. Většina parametrů v této zprávě jsou dynamické. SIPp si je přebírá z přepínačů při startu nástroje nebo z *CSV* souboru. Tyto parametry lze poznat podle hranatých závorek, ve kterých jsou zapsané. Význam těchto parametrů si následně popíšeme:

- *local_ip* – toto pole obsahuje IP adresu počítače, na kterém je SIPp spuštěno
- *field0* – toto pole odkazuje na první sloupec v *CSV* souboru, ze které si načítá názvy účtů
- *transport* – určuje použitý protokol transportní vrstvy (UDP, TCP, TLS)
- *local_port* – určuje číslo portu, standardně port 5060
- *branch* – zde se nachází identifikátor transakce vygenerovaný SIPp
- *call_id* – je identifikátor dialogu, opět vygenerovaný SIPp
- *field1* – toto pole odkazuje na druhý sloupec v *CSV* souboru, odkud si načítá uživatelské jméno a heslo z *CSV* souboru, a představuje celou hlavičku pro autorizaci

Do tagu *recv* zapisujeme kód zprávy odpovědi, kterou má SIPp očekávat. Každá očekávaná odpověď se zapisuje zvlášť do vlastní párové značky s možností označení zprávy jako *optional*. Tento parametr říká, že tato zpráva může, ale také nemusí přijít. V případě, že přijde neočekávaná odpověď, SIPp tuto zprávu vyhodnotí jako chybu a předčasně ukončí scénář. Do odpovědi s kódem 401 nebo 407 se přidává parametr *auth=true*, aby SIPp věděl, že má následující zpráva obsahovat informace nutné pro autentizaci. Parametr *rtd="register"* v posledním *recv* tagu zastavuje časovač, který byl spuštěný na začátku scénáře.

```
<recv response="200" optional="true"></recv>
<recv response="401" auth="true"></recv>
```

Tagy *ResponseTimeRepartition a CallLengthRepartition* definují intervaly, proti kterým se bude interní časovač porovnávat. Ve výsledných statistikách nebudou obsažené přesné časy trvání jednotlivých hovorů, ale počet hovorů, které budou spadat do definovaných intervalů. Výsledky tedy budou například obsahovat 10 hovorů v intervalů 0 až 10 ms, dalších 10 hovorů mezi 10 až 20 ms apod. Logicky platí čím menší intervaly, tím přesnější budou výsledné statistiky.

```
<recv response="200" rtd="register"></recv>
<ResponseTimeRepartition value="10, 20, 30, 40, 50"/>
<CallLengthRepartition value="10, 20, 30, 40, 50"/>
```

Kromě scénáře pro registraci uživatelů na SIP proxy, bude mít tento server druhý scénář, který bude přijímat příchozí hovory. Konfigurace tohoto scénáře probíhá na stejném principu jako scénář pro registraci. Avšak jsou zde pole, které jsou doplňovány na základě probíhající výměny SIP zpráv. Typicky se jedná o pole *From, To, Via* nebo *Route*.

6.2.2 UAC

SIPp v roli UAC bude obsahovat scénáře generující hovory. Tvorba tohoto scénáře probíhá analogicky jako ve výše zmíněném scénáři registrace. Nově se zde, však objevují tagy pro úkony, které se neprovádí na úrovni SIP signalizace. Tyto úkony se zapisují do *action* tagu, které navíc obklopuje tag *nop*. Využít je můžeme pro odesílaní zvukových dat, vyhledávání v záhlaví SIP zprávy, generování unikátního identifikátoru relace pro protokol SDP apod.

Pro náš konkrétní scénář bude důležité provádět vyhledávání v záhlaví a odesílání zvukových dat. Ukázka následujícího kódu provádí prohledávání pole *Contact* nezbytné pro testování SIP proxy. V případě, že hodnota v poli odpovídá regulárnímu výrazu, je tato hodnota přiřazená do proměnné označené jako „1“.

```
<nop>
<action>
<exec regexp="sip:.*[^>]" search_in="hdr" header="Contact:"
check_it="true" assign_to="1" />
</action>
</nop>
```

6.3 Realizace testů

K ověření výkonu celého návrhu budeme testovat množství současných hovorů, které servery zvládnou obsloužit. Server v roli UAC bude pomocí nástroje SIPp generovat hovory trvající 60 sekund. Na druhém serveru bude SIPp v roli UAS přijímat příchozí hovory společně s RTP pakety, přičemž tyto pakety bude vracet zpět ke zdroji. Jednotlivé testy budou probíhat po dobu 5 minut. První minuta testů bude sloužit pro dosažení požadovaného množství současně generovaných hovorů. Sledovat hardwarové vytížení serverů Kamailia a Asterisku budeme po celou dobu trvání testů. Dále budeme sbírat data o kvalitě hovoru v podobě jitteru a zpoždění mezi pakety. Z důvodu limitace nástroje SIPp nebude zapnuto šifrovaní signalizace ani hlasových dat. V případě šifrování signalizace SIPp neumožňuje komunikaci odesílat z více IP adres. K šifrování hlasových dat je nutný mechanismus pro vyjednání šifrovacích klíčů. SIPp tento mechanismus neobsahuje a jedinou možnosti, jak SRTP pakety odesílat je pomocí *pcap* souboru. Tímto způsobem však nelze otestovat dopad na výkon způsobeny šifrováním. Topologie pro měření je shodná s topologii zobrazenou na obrázku 5.1.

Před testováním je nutné zaregistrovat jednotlivé uživatele na Kamailio server. To provedeme níže uvedeným příkazem. Tento příkaz se bude pro UAC a UAS lišit pouze v přepínači „-t“. Tento přepínač zajistí, že budou hovory odesílané z různých IP adres. Při registraci UAS je možné přepínač vynechat. Popis ostatních přepínačů je uveden v odrážkách pod příkazem.

```
sipp -sf uac-register.xml -inf uac-user.csv 10.0.50.65 -r 2000
-m 4000 -t ui
```

- -sf – odkaz na soubor se scénářem
- -inf – odkaz na soubor s informacemi o uživatelích
- -r – množství generovaných zpráv za jednu sekundu
- -m – maximální množství zpráv

Po úspěšně provedených registracích můžeme začít s výkonnostním testem. Nejprve je potřeba spustit SIPp server, tedy UAS.

```
sipp -sf uas-invite.xml -inf uas-user.csv 10.0.50.65 -nd -rtp_echo
```

Přepínače jsou podobné jako u registrace. Nově však přibyl přepínač „-nd“, který zajistí pokračování testu, i v případě přijetí chybové či neočekávané zprávy. Dále přepínač „rtp_echo“ provádějící vracení RTP paketů zpět ke zdroji.

Pro zajištění synchronizace mezi testováním a sběrem metrik budou následující příkazy součástí skriptu. Z důvodu, že jsme se na začátku rozhodli dedikovat vlastní virtuální stroj pro každý nástroj a software, je nutné využít protokolu SSH, kterým se na servery budeme připojovat a spouštět jednotlivé nástroje. SIPp generující hovory budeme pouštět následujícím příkazem:

```
sipp -sf xml/g711a-g711a.xml -inf users-csv/uac-user.csv -t ui \
10.0.50.65 -rp 2s -r $rate -m $max -skip_rlimit -trace_err \
-trace_screen -timer_resol 50 -nd
```

Logika příkazu je shodná s předchozími. Jediným rozdílem jsou přepínače, které přijímají proměnné měnící se podle množství generovaných hovorů. Dále jsou zde přepínače pro uložení dat z měření do textových souborů.

Jako první nástroj pro sběr metrik budeme používat *SAR*. Tento nástroj umožňuje v intervalech zaznamenávat hardwarové vytížení daného serveru. *SAR* je ve většině linuxových distribucí součástí systémových nástrojů pod názvem *SYSTAT*. Instalaci tedy můžeme jednoduše provést přes správce balíčku dané distribuce. Níže uvedeným příkazem na serverech spustíme *SAR* na pozadí, přičemž bude po dobu 5 minut v intervalu 10 sekund sbírat data o vytížení procesoru, paměti a síťového rozhraní. Po ukončení sběru budou data uložena do binárního souboru, jehož název bude obsahovat množství současných hovorů, které jsme v testu generovali.

```
ssh $asterisk_host sudo sar -u -r -P ALL -n DEV \
-o ~/stats/asterisk_${ans}_cc.sar 10 36 > /dev/null 2>&1
```

Jelikož jsou data ukládána do binární podoby, bude je nutné převést do čitelné podoby. To zajistíme pomocí nástroje *sadf*. Ten již není třeba instalovat, jelikož je součástí nástrojů *SYSTAT*. Prvním z níže uvedených příkazů uložíme nasbíraná data o procesoru do *CSV* souboru. Druhým příkazem ukládáme data o paměti a síťového rozhraní do téhož souboru.

```
ssh $asterisk_host "sadf -P ALL -d asterisk_${ans}_cc.sar \
> asterisk_${ans}_cc.csv"
ssh $asterisk_host "sadf -d asterisk_${ans}_cc.sar -- -r -n DEV \
>> asterisk_${ans}_cc.csv"
```

Statistiky RTP paketů budeme sbírat pomocí nástroje *TCPDUMP*. Opět lze tento nástroj nainstalovat pomocí správce balíčků. Metriky budeme sbírat na serveru s UAS, jakožto koncový účastník.

```
ssh $uas_host sudo timeout 360 tcpdump -i enp0s3 \
-w ~/stats/rtp_metrics_${ans}_cc.pcap > /dev/null 2>&1
```

Obsah celého skriptu bude umístěn do příloh. Zde mohou případní zájemci prozkoumat pořadí jednotlivých příkazu nebo způsob, jakým jsou proměnné testu vypočítávány.

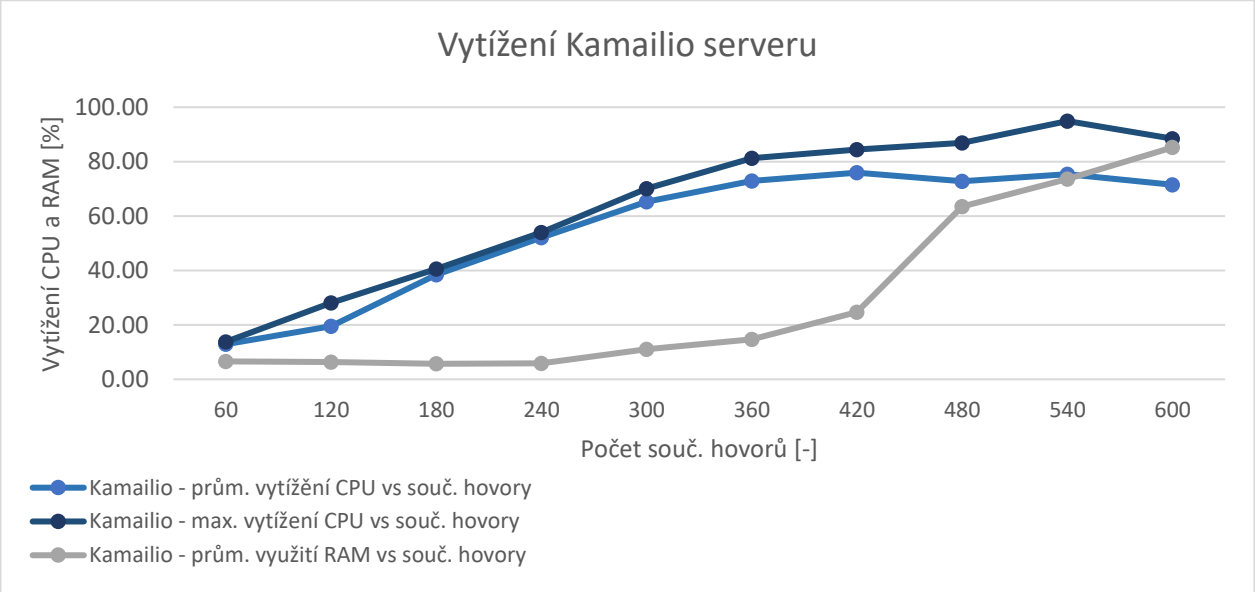
Testování budeme začínat na 60 současně generovaných hovorech. Zatížení budeme postupně zvyšovat, dokud nenarazíme na limit jednoho ze serveru. Jako kodek pro hlasová data využijeme *G.711 A-law* a *G.726*.

6.4 Analýza naměřených výsledků

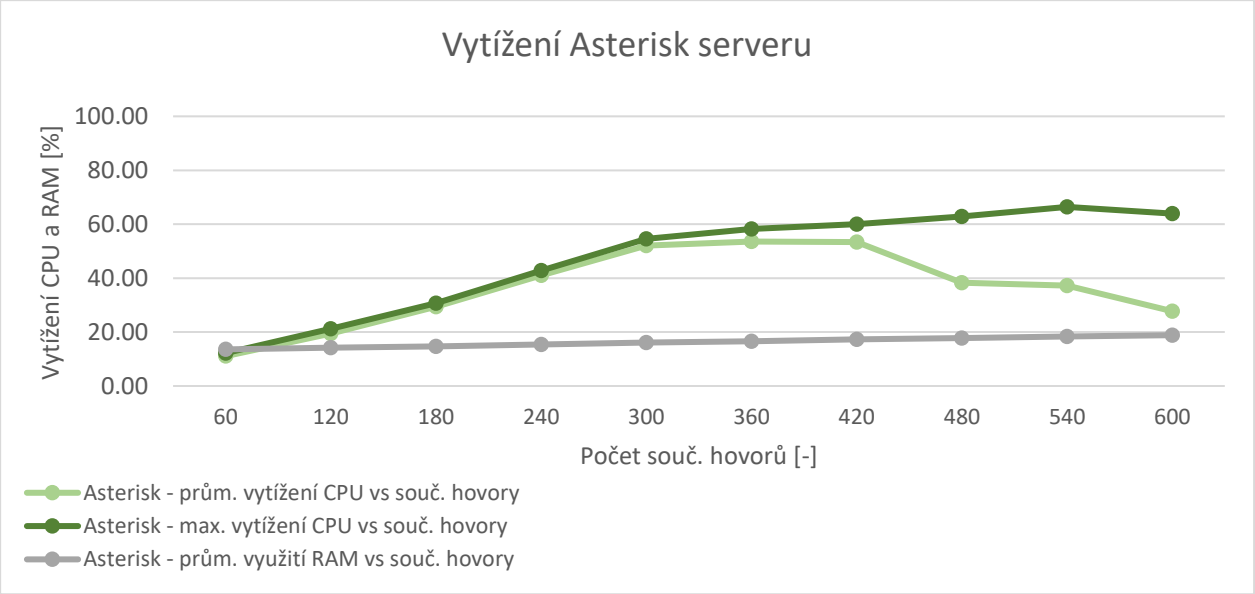
Před vyhodnocením výsledků je nutné brát v úvahu, že metriky serveru byly sbírané po celý proces daného testu. Tedy i v době, kdy počet současných hovorů nebyl konstantní. Proto je nezbytné z nasbíraných dat vyfiltrovat pouze část, kde počet současně generovaných hovorů dosahoval požadovaného maxima. Poté lze hodnoty z vyfiltrované části použít k získaní průměrné hodnoty vytížení procesoru a operační paměti serverů. K analýze kvalitativních metrik poslouží program Wireshark. Prostřednictvím tohoto programu budou vybrané hodnoty jitteru a zpoždění mezi pakety z hovoru, který probíhal v nejvyšší zátěži.

6.4.1 G.711 A-law – G.711 A-law

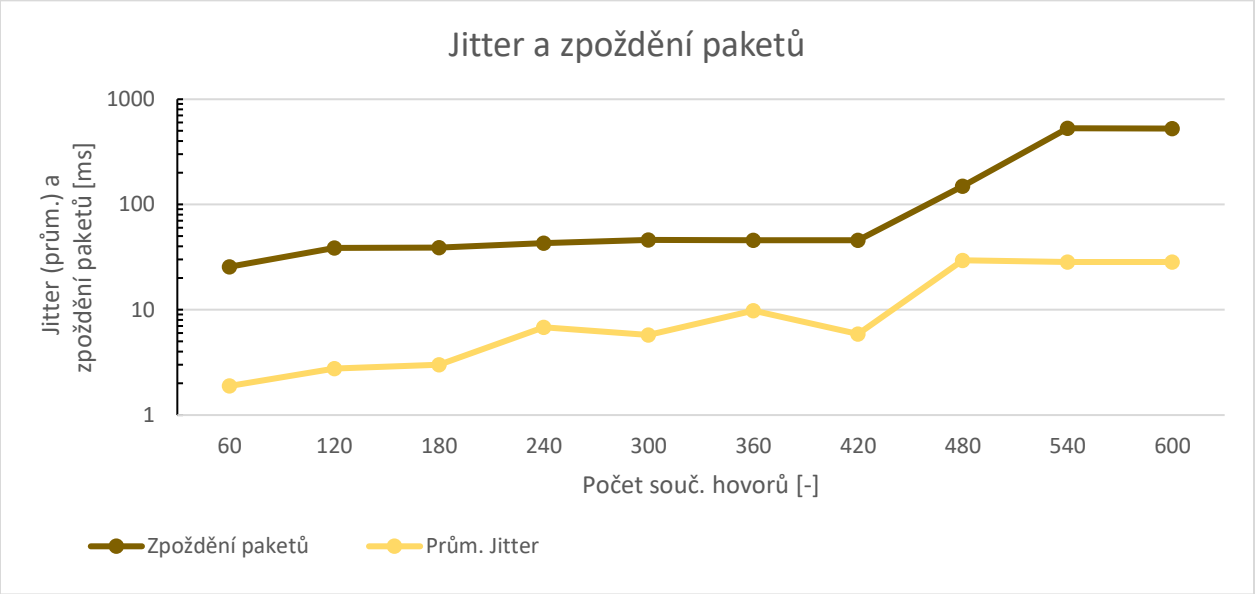
První dva grafy zobrazují vytížení procesoru a operační paměti na obou serverech. Z grafů lze vidět, že jsme narazili na první limit u serveru s Kamailiem. Na tomto serveru vytížení bylo způsobeno především RTPProxy, přičemž byl tento nástroj schopný zpracovat 420 současných hovorů. Při větším množství už byl procesor natolik zatížený, že docházelo i k zahlcení paměti. Mezitím průměrné vytížení procesoru na Asterisk serveru začalo klesat, což bylo způsobeno neschopnosti Kamailia zpracovávat další požadavky. Další dva grafy znázorňují kvalitativní metriky a počet neúspěšných hovorů v jednotlivých testech. Při 480 současně generovaných hovorech a více dochází ke značnému zhoršení kvality hovorů, a ukazuje se tady i první neúspěšný hovor.



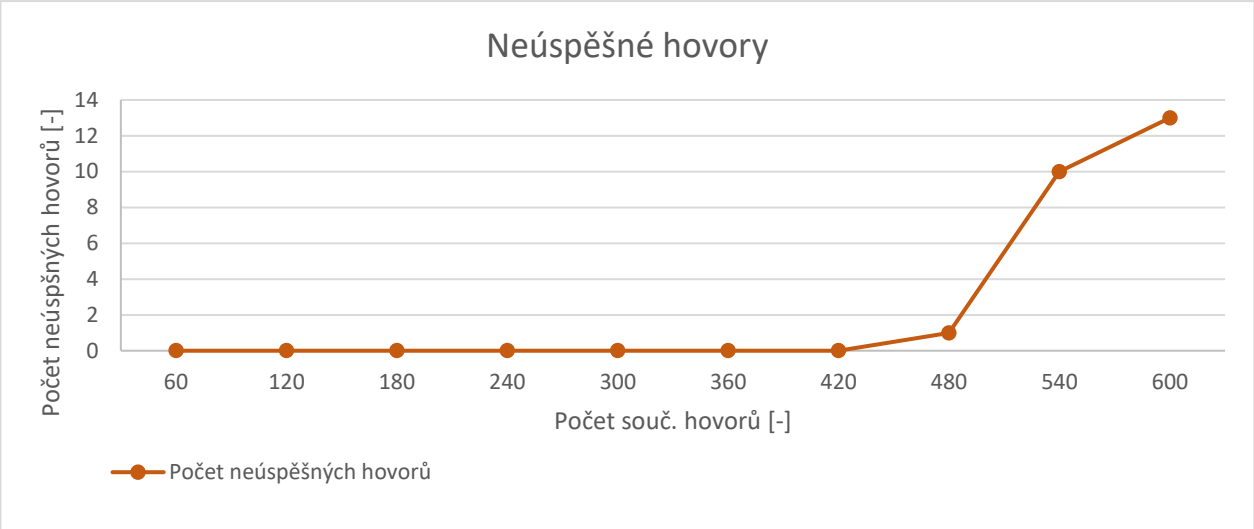
Obr. 6.1: G.711A-G.711A - vytížení Kamailio serveru



Obr. 6.2: G.711A-G.711A - vytížení Asterisk serveru



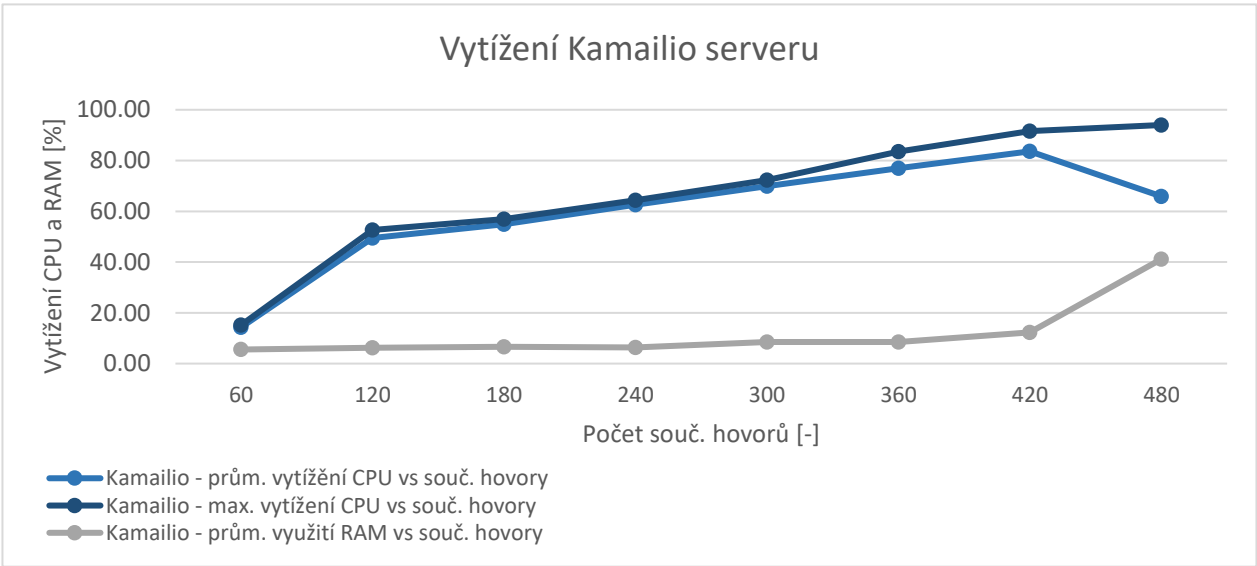
Obr. 6.3: G.711A-G.711A - jitter a zpoždění hovorů



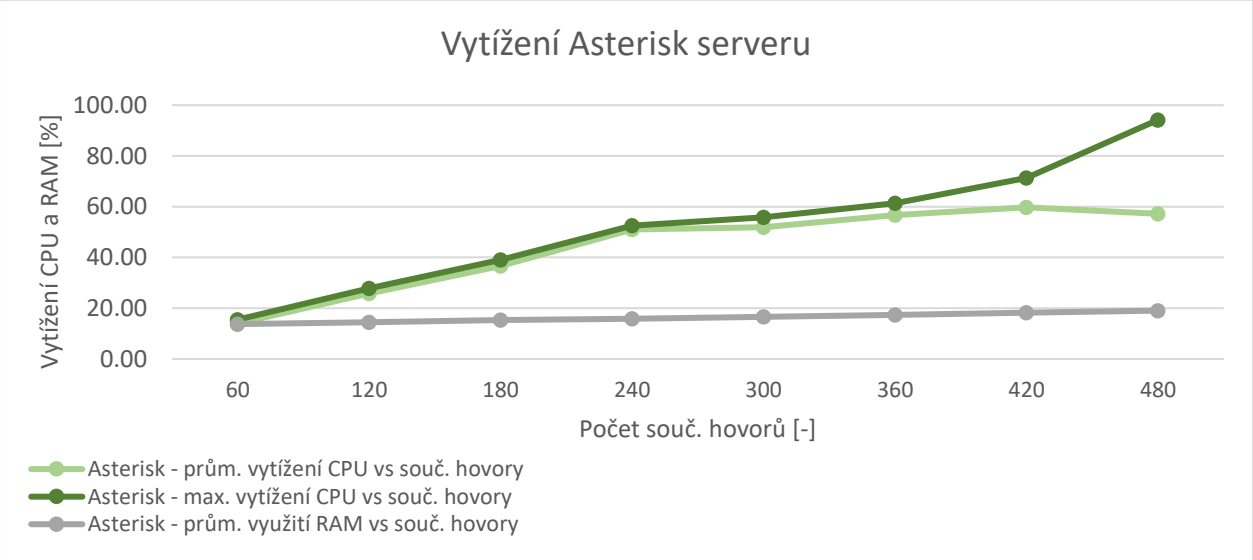
Obr. 6.4: G.711A-G.711A - neúspěšné hovory vs souč. hovory

6.4.2 G.726 – G.711 A-law

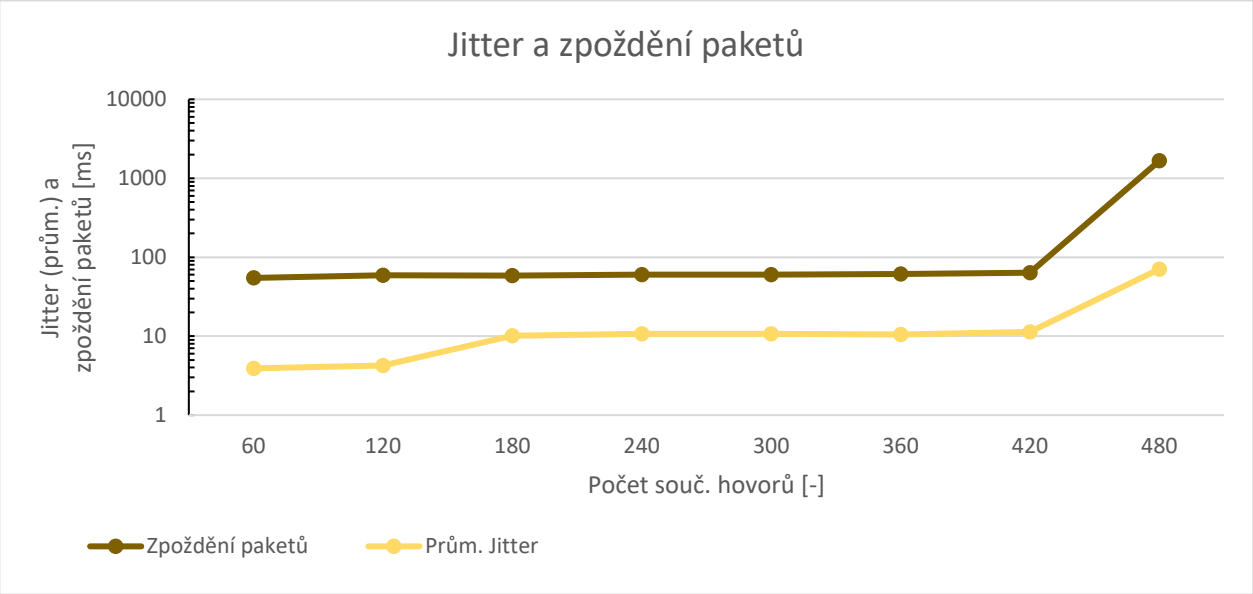
Při tomto testu docházelo ke translaci mezi kodekem *G.726* a *G.711 A-law*, což se projevilo na vyšším průměrném zatížení procesoru Asterisku. Dále lze pozorovat mírné zhoršení ve kvalitě hovorů oproti předchozímu testu. Počet současných hovorů, které servery byly schopné zpracovat zůstal stejný, tedy 420. Při testování 480 současně generovaných hovorů již docházelo k restartům RTPProxy, proto vyšší zatížení nebylo testováno. Následek restartů se také projevil u výsledné hodnoty průměrného zatížení procesoru serverů.



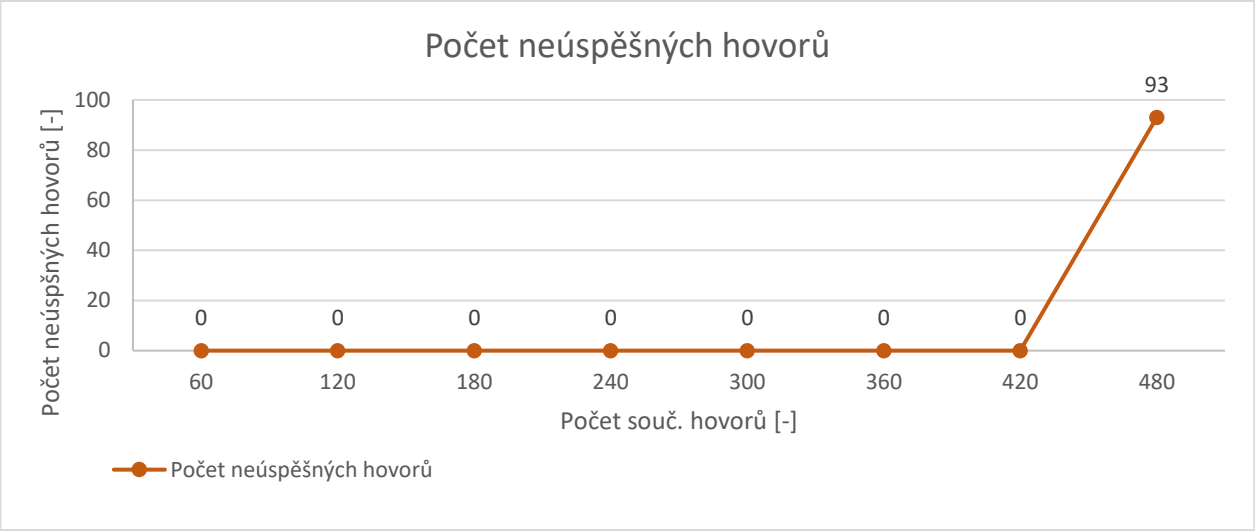
Obr. 6.5: G.726-G.711A - vytížení Kamailio serveru



Obr. 6.6: G.726-G.711A - vytížení Asterisk serveru



Obr. 6.7: G.726-G.711A - jitter a zpoždění



Obr. 6.8: G.726-G.711A - neúspěšné hovory vs souč. hovory

7 Závěr

Cílem této práce byla implementace bezpečnostních funkcí pomocí open source softwarů Asterisk a Kamailio. Předtím však bylo nutné zajistit smysluplnou komunikaci mezi oběma softwary. Z tohoto hlediska měl Kamailio na starost především SIP signalizaci, registraci uživatelů a jejich lokalizaci. Asterisk následně zajistil spojení hovorů, šifrování RTP paketů, a případně translaci mezi kodeky.

První implementovaná funkce byla maskování topologie. Účelem bylo nechat viditelný pouze Kamailio server, jakožto SIP proxy a ostatní zamaskovat. Nezbytné tedy bylo zajistit maskování uživatelů a serveru s Asteriskem. Maskování jednotlivých účastníků zajistil Asterisk v režimu B2BUA. Samotné maskování Asterisku bylo dosaženo kombinaci modulu *topoh* z Kamailia a nástroje RTPProxy. Modul *topoh* se staral o maskování informací o Asterisku v SIP signalizaci, zatímco RTPProxy kontroloval tok RTP paketů. Výhodou modulu *topoh* je, že při maskování nemaže žádná pole ze SIP záhlaví. Tudíž maskování pomocí tohoto modulu nemá vliv na směrování jednotlivých zpráv. Další výhodou je použití RTPProxy, který mimo maskování také umožňuje určitou škálovatelnost daného návrhu.

Dále, pomocí modulu *htable* a *pike*, byla implementována ochrana proti DoS útokům. Tyto dva moduly se starají o blokování IP adres, ze kterých přichází velké množství požadavků v krátkém časovém intervalu. Dále bylo vysvětleno, proč pomocí těchto modulů nelze zajistit ochranu proti DDoS útokům, a jak lze alespoň částečně těmto útokům předcházet pomocí *GeoIP* modulu.

Ochrana proti útokům Man in the middle, odposlouchávání a dalším typům DoS útoků, které spoléhají na zachycení nezabezpečené komunikace, byla zajištěna na straně Kamailia šifrováním signalizace pomocí TLS protokolu a na straně Asterisku šifrováním mediálních dat pomocí SRTP protokolu.

Proti SQL injekcí byla implementována funkce z modulu *secfilter*. Tato funkce provádí kontrolu nelegálních znaků v polích *User-agent*, *From*, *To* a *Contact* záhlaví SIP zprávy. Následně byla dopsána doplňující kontrola pro pole *Authorization*.

O správné formátování SIP zpráv se stará *sanity* modul, který je již obsažen ve výchozí konfiguraci Kamailia. V této části je tedy pouze popsána funkce tohoto modulu.

Nakonec jsou implementované bezpečnostní mechanismy, jejichž úkolem bylo zamezit skenování platných účtů nebo hádání hesel. Dále povolení určitých služeb pouze registrovaným uživatelům a zavedení černé listiny nežádoucích uživatelů.

Po úspěšné implementaci bezpečnostních funkcí následovalo otestování výkonu navrženého řešení. Pro testování byl využit nástroj SIPp, který umožňuje tvorbu vlastních scénářů hovorů. Jako vyhodnocovací metrika byl vybrán počet současně generovaných hovorů, které servery byly schopné zpracovat. Celkově byly vytvořené dva testovací scénáře. Jeden scénář zatěžoval servery hovory bez nutnosti translace mezi kodeky, zatímco druhý scénář testoval efektivitu translace mezi kodeky *G.726* a *G.711 A-law*.

Finální výsledky měření ukazovaly, že v obou případech je navržené řešení schopné zpracovat až 420 současně generovaných hovorů. Při větším množství již docházelo k neúspěšným hovorům. V obou testech nejvíce výkonu vyžadoval nástroj RTPProxy, který byl schopen server zatížit až do míry, kdy

Kamailio nedokázal zpracovat některé z příchozích hovorů. Zatímco zatížení procesoru na serveru s Asteriskem se pohybovalo maximálně okolo 60 %. Z pohledu škálovatelnosti se jednalo o nemilé zjištění, jelikož ji měl zajišťovat právě RTPProxy. Jedno z možných vysvětlení je, že tento nástroj ve virtualizovaném prostředí nedosahuje optimálního výkonu. Toto tvrzení bohužel nebylo možné s jistotou potvrdit, jelikož druhý stroj s identickými parametry nebyl k dispozici. I přes tento fakt byl proveden dodatečný test bez translace mezi kodeky pro 600 současně generovaných hovorů. V tomto testu byl RTPProxy nainstalován na notebook s dvoujádrovým procesorem (i5-5300U) a operačním systémem Ubuntu 19.04. Naměřené výsledky jsou shrnuté v následující tabulce:

	600CC	**NTB – 600CC**	
	Kamailio + RTPProxy	**Kamailio**	**NTB s RTPProxy**
CPU prům. [%]	71.55	4.72	57.85
CPU max. [%]	88.42	5.46	61.52
RAM prům. [%]	85.22	2.81	13.21

Tab. 2 Srovnání testů

Z tabulky lze vyčíst, že 600 současně generovaných hovorů nebyly schopné notebook plně vytížit, což potvrzuje i nulový počet neúspěšných hovorů. Je nutné podotknout, že toto srovnání musíme brát s určitou rezervou. Dochází totiž ke srovnávání dvou různých procesorů s odlišnými výrobci i architekturou. Toto je také důvod, proč nejsou tyto výsledky umístěné v hlavní části práce. Nicméně budou uvedené v přílohách společně s ostatními výsledky testování.

Výsledný návrh ukazuje jednu z možností realizace bezpečnostního řešení pro SIP protokol pomocí open source softwarů. V návrhu je brán ohled na škálovatelnost výsledného řešení pomocí RTPProxy. V ideálním případě by měl být tento nástroj schopen obsluhovat RTP tok z více Asterisk serverů, mezi nimiž by se provádělo vyvažování zátěže. Tento návrh lze také využít jako základ pro realizaci komplexního SBC prvku, nebo k němu přidat další hlasové služby, jako například interaktivní hlasový systém nebo konferenční server. Celkově se tedy jedná o poměrně flexibilní řešení.

Použitá literatura

[1] MALINA, Lukáš a Václav ZEMAN. Comprehensive Security in SIP. Elektrorevue. 2011, (Vol. 2, 1), 7-14. ISSN 1213-1539. Dostupné z: http://elektrorevue.cz/file.php?id=200000710-dde8fdee2f

[2] JOHNSTON, Alan B. SIP: understanding the Session Initiation Protocol. 3rd ed. Boston: Artech House, c2009. ISBN 978-1-60783-995-8.

[3] ŠAFAŘÍK, Jakub. Zvýšení odolnosti SIP Proxy proti DoS útokům. Ostrava, 2011. Diplomová práce. VŠB - Technická univerzita Ostrava.

[4] Oracle Communications Session Border Controller Security Guide. Release S-Cz8.3.0 - for Service Provider and Enterprise [online]. 2019 [cit. 2020-03-20]. Dostupné z: https://docs.oracle.com/cd/F12246_01/html/sbc_scz830_security/GUID-1AAADEA4-DB4E-4D95-916C-2EA9AE7DD7B5.htm

[5] CAMARILLIO, Gonzalo, Robert F. PENFIELD, Alan HAWRYLYSHEN a MEdhavi BHATIA, HAUTAKORPI, Jani, ed. *Requirements from Session Initiation Protocol (SIP) Session Border Control (SBC) Deployments*, RFC 5853, Duben 2010 [online]. [cit. 2020-03-19]. ISSN 2070-1721. Dostupné z: https://tools.ietf.org/html/rfc5853

[6] DAVENPORT, Malcom a Joshua C. COLP. A Brief History of the Asterisk Project. Asterisk Wiki [online]. 2019 [cit. 2020-05-13]. Dostupné z: https://wiki.asterisk.org/wiki/display/AST/A+Brief+History+of+the+Asterisk+Project

[7] BRYANT, Russell. Asterisk: the definitive guide. Fourth edition. Sebastopol: O'Reilly, [2013]. ISBN 978-1-449-33242-6.

[8] Asterisk Applications [online]. [cit. 2020-01-10]. Dostupné z: https://www.asterisk.org/get-started/applications

[9] VOZŇÁK, Miroslav. Technologie a protokoly multimediálních komunikací pro integrovanou výuku VUT a VŠB-TUO. Ostrava: Vysoká škola báňská - Technická univerzita Ostrava, 2014. ISBN 978-80-248-3326-2.

[10] DAVENPORT, Malcom a Joshua C. COLP. Creating Dialplan Extensions. Asterisk Wiki [online]. 2019 [cit. 2020-01-12]. Dostupné z: https://wiki.asterisk.org/wiki/display/AST/Creating+Dialplan+Extensions

[11] BRIGHT, Sean a Corey FARRELL. Asterisk Best Practices. GitHub [online]. 2019 [cit. 2020-01-12]. Dostupné z: https://github.com/asterisk/asterisk/blob/master/README-SERIOUSLY.bestpractices.md

[12] KRČMÁŘ, Petr. Fail2ban: konec hádání hesel na serveru. Root.cz [online]. 2013 [cit. 2020-04-24]. Dostupné z: https://www.root.cz/clanky/fail2ban-konec-hadani-hesel-na-serveru/

[13] MIERLA, Daniel-Constantin a Elena-Ramona MODROIU. Kamailio SIP Server v3.2.0 Development Guide. Asipto.com [online]. 2011 [cit. 2020-03-18]. Dostupné z: http://www.asipto.com/pub/kamailio-devel-guide/#c05datastruct

[14] ROZHON, Jan. Výkonnostní testování SIP infrastruktury. Ostrava, 2010. Diplomová práce. VŠB. Vedoucí práce Doc. Ing. Miroslav Vozňák, Ph.D.

[15] MODROIU, Elena-Ramona, BALASHOV, Alex a Ovidiu SAS, ed. HTable Module. Kamailio.org [online]. 2020 [cit. 2020-01-09]. Dostupné z: https://kamailio.org/docs/modules/5.3.x/modules/htable.html

[16] TIRPAK, Miklos a Emmanuel SCHMIDBAUER, IANCU, Bogdan-Andrei a Juha HEINANEN, ed. Permissions Module. Kamailio.org [online]. 2020 [cit. 2020-01-09]. Dostupné z: https://www.kamailio.org/docs/modules/5.3.x/modules/permissions.html

[17] IANCU, Bogdan-Andrei. Pike Module. Kamailio.org [online]. 2020 [cit. 2020-01-09]. Dostupné z: https://www.kamailio.org/docs/modules/5.3.x/modules/pike.html

[18] MIERLA, Daniel-Constantin a Hendrik SCHOLZ, SAS, Ovidiu, ed. Pipelimit Module. Kamailio.org [online]. 2020 [cit. 2020-01-09]. Dostupné z: https://www.kamailio.org/docs/modules/5.3.x/modules/pipelimit.html

[19] SAS, Ovidiu, Bogdan Vasile HARJOC a Hendrik SCHOLZ. Ratelimit Module. Kamailio.org [online]. 2020 [cit. 2020-01-09]. Dostupné z: https://www.kamailio.org/docs/modules/5.3.x/modules/ratelimit.html

[20] OHLMEIER, Nils. The Sanity Module - SIP syntax checking, Kamailio.org [online]. 2020 [cit. 2020-01-09]. Dostupné z: https://www.kamailio.org/docs/modules/5.3.x/modules/sanity.html

[21] LUIS VERDEGUER, Jose. Secfilter Module. Kamailio.org [online]. 2020 [cit. 2020-01-09]. Dostupné z: https://www.kamailio.org/docs/modules/5.3.x/modules/secfilter.html

[22] MIERLA, Daniel-Constantin. Topoh Module. Kamailio.org [online]. 2020 [cit. 2020-01-09]. Dostupné z: https://www.kamailio.org/docs/modules/5.3.x/modules/topoh.html

[23] GAYRAUD, Richard, Olivier JACQUES a další. SIPp reference documentation. SIPp [online]. 2014 [cit. 2020-04-24]. Dostupné z: http://sipp.sourceforge.net/doc3.0/reference.html

[24] ŘEZÁČ, FIlip, Miroslav VOZŇÁK a Jan ROZHON. Bezpečnost v komunikacích. Ostrava, 2013. Vysoká škola báňská - Technická univerzita Ostrava.

Seznam příloh

Tištěné přílohy

Elektronické přílohy

PŘÍLOHA E: KONFIGURAČNÍ SOUBORY ASTERISKU
PŘÍLOHA F: KONFIGURAČNÍ SOUBORY KAMAILIO
PŘÍLOHA G: SCÉNÁŘE SIPP
PŘÍLOHA H: VÝSLEDKY MĚŘENÍ

Příloha A: Instalace Asterisku

V prvním kroku stáhneme balíček se zdrojovými kódy Asterisku a následně jej rozbalíme.

```
$ curl -O https://downloads.asterisk.org/pub/telephony/asterisk/
  asterisk/asterisk-16-current.tar.gz
$ tar xvf asterisk-16-current.tar.gz
$ cd asterisk-16.6.2
```

Dále doinstalujeme závislostí.

```
$ ./contrib/scripts/install_prereq
$ ./configure
```

Pomocí následujících příkazů vyvoláme menu s výběrem modulů, které chceme zkompilovat, a zkompilujeme Asterisk.

```
$ make menuselect
$ make
$ make install
$ make config
$ ldconfig
```

Nezbytné je také vytvořit uživatele asterisk a přidělit mu práva do složek, se kterými Asterisk pracuje.

```
$ groupadd asterisk
$ useradd -r -d /var/lib/asterisk -g asterisk asterisk
$ usermod -aG audio,dialout asterisk
$ chown -R asterisk:asterisk /etc/asterisk /var/{lib,log,spool}/asterisk
  /usr/lib/asterisk
```

Nakonec nastavíme program, aby se spouštěl pod tímto uživatelem.

```
$ nano /etc/default/asterisk
    AST_USER="asterisk"
    AST_GROUP="asterisk"
$ nano /etc/asterisk/asterisk.conf
     runuser = asterisk
     rungroup = asterisk
$ sudo systemctl restart asterisk
$ sudo systemctl enable asterisk
```

Příloha B: Instalace Kamailia

Instalaci začínáme doinstalováním závislosti a MySQL serveru.

```
$ apt install git git-core gcc g++ flex bison libmysqlclient-dev default
  libmysqlclient-dev make autoconf libssl-dev libcurl4-openssl-dev
  libxml2-dev libpcre3-dev mysql-server
```

Dále vytvoříme adresář, kde budou zdrojové kódy Kamailia uloženy a stáhneme jej:

```
$ mkdir -p /usr/local/src/kamailio-5.3
$ cd /usr/local/src/kamailio-5.3
$ git clone --depth 1 --no-single-branch https://github.com/kamailio/
  kamailio kamailio
$ cd kamailio
$ git checkout -b 5.3 origin/5.3
```

Kamailio zkompilujeme i s moduly pro MySQL a TLS.

```
$ make include_modules="db_mysql tls" cfg
$ make all
$ make install
$ make install-systemd-debian
```

Příloha C: Vytvoření certifikační autority

Pro certifikační autoritu vytvoříme potřebné adresáře a soubory.

```
$ mkdir -p ~/.demoCA
$ cd ~/.demoCA
$ mkdir newcerts
$ mkdir private
$ chmod 0700 ~/.demoCA
$ touch index.txt
$ echo 00 > serial
```

Nyní vytvoříme samotnou certifikační autoritu.

```
$ openssl req -config /etc/ssl/openssl.cnf -new -x509 -out ca.crt -keyout
  ~/.demoCA/private/ca.key -days 3650 -newkey rsa:4096
$ chmod 0400 private/ca.key
```

Příloha D: Instalace SIPp

Pro SIPp nainstalujeme následující závislosti.

```
$ apt install gcc g++ libncurses-dev openssl libssl-dev libpcap0.8-dev
  libnet1-dev libgsl0-dev
```

Následně stáhneme a zkompilujeme balíček se zdrojovým kódem nástroje.

```
$ wget https://github.com/SIPp/sipp/releases/download/v3.6.0/sipp-3.6.0
  .tar.gz
$ tar xvf sipp-3.6.0.tar.gz
$ cd sipp-3.6.0/
$ ./configure --with-openssl --with-pcap
$ make
```

Příloha E: Asterisk – extensions.conf

```
[kamailio]
exten => _[0-9a-zA-Z].,1,Dial(PJSIP/kamailio/sip:${EXTEN}@${SIPDOMAIN},
10)
```

Příloha F: Asterisk – pjsip.conf

```
;=================GLOBAL PARAMETERS==========
[global]
type = global
user_agent = Some generic B2BUA
;=================TRANSPORTS=================
[udp-transport]
type = transport
protocol = udp
bind = 10.0.50.55

[tcp-transport]
type = transport
protocol = tcp
bind = 10.0.50.55
;==================KAMAILIO===============
[kamailio]
type = endpoint
context = kamailio
disallow = all
```

```
allow = alaw
allow = g726
direct_media = no
from_domain = 10.0.50.65
contact_user = kamailio
media_encryption = sdes
media_encryption_optimistic = yes

[kamailio]
type = identify
match = 10.0.50.65
endpoint = kamailio
```

Příloha G: kamailio.cfg

```
#!KAMAILIO
#!define WITH_MYSQL
#!define WITH_AUTH
#!define WITH_IPAUTH
#!define WITH_TLS
#!define WITH_ASTERISK
#!define WITH_ANTIFLOOD
#
#!ifdef ACCDB_COMMENT
ALTER TABLE acc ADD COLUMN src_user VARCHAR(64) NOT NULL DEFAULT '';
ALTER TABLE acc ADD COLUMN src_domain VARCHAR(128) NOT NULL DEFAULT '';
ALTER TABLE acc ADD COLUMN src_ip varchar(64) NOT NULL default '';
ALTER TABLE acc ADD COLUMN dst_ouser VARCHAR(64) NOT NULL DEFAULT '';
ALTER TABLE acc ADD COLUMN dst_user VARCHAR(64) NOT NULL DEFAULT '';
ALTER TABLE acc ADD COLUMN dst_domain VARCHAR(128) NOT NULL DEFAULT '';
ALTER TABLE missed_calls ADD COLUMN
src_user VARCHAR(64) NOT NULL DEFAULT '';
ALTER TABLE missed_calls ADD COLUMN
src_domain VARCHAR(128) NOT NULL DEFAULT '';
ALTER TABLE missed_calls ADD COLUMN
src_ip varchar(64) NOT NULL default '';
ALTER TABLE missed_calls ADD COLUMN
dst_ouser VARCHAR(64) NOT NULL DEFAULT '';
ALTER TABLE missed_calls ADD COLUMN
dst_user VARCHAR(64) NOT NULL DEFAULT '';
ALTER TABLE missed_calls ADD COLUMN
```

```
dst_domain VARCHAR(128) NOT NULL DEFAULT '';
#!endif

####### Include Local Config If Exists #########
import_file "kamailio-local.cfg"

####### Defined Values #########

#!ifdef WITH_MYSQL
#!ifndef DBURL
#!define DBURL"mysql://kamailio:StrongPassword@localhost/kamailio"
#!endif
#!endif
#!ifdef WITH_MULTIDOMAIN
# - the value for 'use_domain' parameters
#!define MULTIDOMAIN 1
#!else
#!define MULTIDOMAIN 0
#!endif

# - flags
#   FLT_ - per transaction (message) flags
#   FLB_ - per branch flags
#!define FLT_ACC 1
#!define FLT_ACCMISSED 2
#!define FLT_ACCFAILED 3
#!define FLT_NATS 5

#!define FLB_NATB 6
#!define FLB_NATSIPPING 7

####### Global Parameters #########

### LOG Levels: 3=DBG, 2=INFO, 1=NOTICE, 0=WARN, -1=ERR
#!ifdef WITH_DEBUG
debug=2
log_stderror=yes
#!else
debug=2
log_stderror=no
#!endif
```

```
memdbg=5
memlog=5
open_files_limit=16384
log_facility=LOG_LOCAL0
log_prefix="{$mt $hdr(CSeq) $ci} "

/* number of SIP routing processes for each UDP socket
 * - value inherited by tcp_children and sctp_children when not set exp
licitely */
children=4

/* uncomment the next line to disable TCP (default on) */
# disable_tcp=yes

/* number of SIP routing processes for all TCP/TLS sockets */
# tcp_children=8

/* uncomment the next line to disable the auto discovery of local alias
es
 * based on reverse DNS on IPs (default on) */
auto_aliases=no

/* add local domain aliases */
# alias="sip.mydomain.com"

/* uncomment and configure the following line if you want Kamailio to
 * bind on a specific interfacetcp_max_connections=2048

#!ifdef WITH_TLS
enable_tls=yes
tls_max_connections=2048
#!endif

####### Custom Parameters #########

#!ifdef WITH_ASTERISK
asterisk.bindip = "10.0.50.55" desc "Asterisk IP address"
asterisk.bindport = "5060" desc "Asterisk Port"
#!endif
```

```
server_signature = no # not to put a server header
sip_warning = 0

tcp_accept_no_cl=yes
######## Modules Section ########

# mpath="/usr/local/lib64/kamailio/modules/"

#!ifdef WITH_MYSQL
loadmodule "db_mysql.so"
#!endif

loadmodule "jsonrpcs.so"
loadmodule "kex.so"
loadmodule "corex.so"
loadmodule "tm.so"
loadmodule "tmx.so"
loadmodule "sl.so"
loadmodule "rr.so"
loadmodule "pv.so"
loadmodule "maxfwd.so"
loadmodule "usrloc.so"
loadmodule "registrar.so"
loadmodule "textops.so"
loadmodule "siputils.so"
loadmodule "xlog.so"
loadmodule "sanity.so"
loadmodule "ctl.so"
loadmodule "cfg_rpc.so"
loadmodule "acc.so"
loadmodule "counters.so"
loadmodule "topoh.so"
loadmodule "rtpproxy.so"
loadmodule "ipops.so"
loadmodule "geoip2"
loadmodule "permissions.so"
loadmodule "secfilter.so"

#!ifdef WITH_AUTH
loadmodule "auth.so"
loadmodule "auth_db.so"
```

```
#!ifdef WITH_IPAUTH
#!endif
#!endif

#!ifdef WITH_ALIASDB
loadmodule "alias_db.so"
#!endif

#!ifdef WITH_SPEEDDIAL
loadmodule "speeddial.so"
#!endif

#!ifdef WITH_MULTIDOMAIN
loadmodule "domain.so"
#!endif

#!ifdef WITH_PRESENCE
loadmodule "presence.so"
loadmodule "presence_xml.so"
#!endif

#!ifdef WITH_NAT
loadmodule "nathelper.so"
#!endif

#!ifdef WITH_TLS
loadmodule "tls.so"
#!endif

#!ifdef WITH_ANTIFLOOD
loadmodule "htable.so"
loadmodule "pike.so"
loadmodule "pipelimit.so"
#!endif

#!ifdef WITH_XMLRPC
loadmodule "xmlrpc.so"
#!endif

#!ifdef WITH_DEBUG
loadmodule "debugger.so"
```

```
#!endif

# ----------------- setting module-specific parameters ---------------

# ----- jsonrpcs params -----
modparam("jsonrpcs", "pretty_format", 1)
modparam("jsonrpcs","fifo_name", "/var/run/kamailio/kamailio_rpc.fifo")
modparam("jsonrpcs","dgram_socket",
"/var/run/kamailio/kamailio_rpc.sock")

# ----- ctl params -----
/* set the path to RPC unix socket control file */
modparam("ctl", "binrpc", "unix:/var/run/kamailio/kamailio_ctl")
modparam("ctl", "binrpc_buffer_size", 4096)
modparam("ctl", "binrpc_max_body_size", 10000)
modparam("ctl", "binrpc_struct_max_body_size", 300000)

# ----- sanity params -----
modparam("sanity", "autodrop", 0)

# ----- tm params -----
# auto-discard branches from previous serial forking leg
modparam("tm", "failure_reply_mode", 3)
# default retransmission timeout: 30sec
modparam("tm", "fr_timer", 30000)
# default invite retransmission timeout after 1xx: 120sec
modparam("tm", "fr_inv_timer", 120000)

# ----- rr params -----
modparam("rr", "enable_full_lr", 0)
modparam("rr", "append_fromtag", 1)
modparam("rr", "force_send_socket", 1)

# ----- topoh params -----
modparam("topoh", "mask_key", "ceecaiHi4Equi7k")
modparam("topoh", "mask_ip", "127.0.0.10")
modparam("topoh", "sanity_checks", 1)

# ----- registrar params -----
modparam("registrar", "method_filtering", 1)
modparam("registrar", "max_contacts", 10)
```

```
modparam("registrar", "min_expires", 300)
modparam("registrar", "max_expires", 86400)
modparam("registrar", "gruu_enabled", 0)

# ----- acc params -----
modparam("acc", "early_media", 0)
modparam("acc", "report_ack", 0)
modparam("acc", "report_cancels", 0)
modparam("acc", "detect_direction", 0)

modparam("acc", "log_flag", FLT_ACC)
modparam("acc", "log_missed_flag", FLT_ACCMISSED)
modparam("acc", "log_extra",
    "src_user=$fU;src_domain=$fd;src_ip=$si;"
    "dst_ouser=$tU;dst_user=$rU;dst_domain=$rd")
modparam("acc", "failed_transaction_flag", FLT_ACCFAILED)
#!ifdef WITH_ACCDB
modparam("acc", "db_flag", FLT_ACC)
modparam("acc", "db_missed_flag", FLT_ACCMISSED)
modparam("acc", "db_url", DBURL)
modparam("acc", "db_extra",
    "src_user=$fU;src_domain=$fd;src_ip=$si;"
    "dst_ouser=$tU;dst_user=$rU;dst_domain=$rd")
#!endif

# ----- usrloc params -----
#!ifdef WITH_USRLOCDB
modparam("usrloc", "db_url", DBURL)
modparam("usrloc", "db_mode", 2)
modparam("usrloc", "use_domain", MULTIDOMAIN)
#!endif

# ----- auth_db params -----
#!ifdef WITH_AUTH
modparam("auth_db", "use_domain", MULTIDOMAIN)
modparam("auth_db", "db_url", DBURL)
modparam("auth_db", "calculate_ha1", yes)
modparam("auth_db", "load_credentials", "")
modparam("auth_db", "user_column", "username")
modparam("auth_db", "password_column", "password")
modparam("auth_db", "version_table", 1)
```

```
# ----- auth params -----
modparam("auth", "nonce_count", 1)
modparam("auth", "qop", "auth")
modparam("auth", "nonce_expire", 60)
modparam("auth", "nonce_auth_max_drift", 2)
#!endif

# ----- permissions params -----
modparam("permissions", "db_url", DBURL)
modparam("permissions", "db_mode", 1)

# ----- alias_db params -----
#!ifdef WITH_ALIASDB
modparam("alias_db", "db_url", DBURL)
modparam("alias_db", "use_domain", MULTIDOMAIN)
#!endif

# ----- speeddial params -----
#!ifdef WITH_SPEEDDIAL
modparam("speeddial", "db_url", DBURL)
modparam("speeddial", "use_domain", MULTIDOMAIN)
#!endif

# ----- domain params -----
#!ifdef WITH_MULTIDOMAIN
modparam("domain", "db_url", DBURL)
modparam("domain", "register_myself", 1)
#!endif

#!ifdef WITH_PRESENCE
# ----- presence params -----
modparam("presence", "db_url", DBURL)

# ----- presence_xml params -----
modparam("presence_xml", "db_url", DBURL)
modparam("presence_xml", "force_active", 1)
#!endif

# ----- rtpproxy params -----
modparam("rtpproxy", "rtpproxy_sock", "unix:/var/run/rtpproxy.sock")
```

```
#modparam("rtpproxy", "rtpproxy_sock", "udp:10.0.50.30:9000")
modparam("rtpproxy", "rtpproxy_retr", 1)

#!ifdef WITH_NAT
# ----- nathelper params -----
modparam("nathelper", "natping_interval", 30)
modparam("nathelper", "ping_nated_only", 1)
modparam("nathelper", "sipping_bflag", FLB_NATSIPPING)
modparam("nathelper|registrar", "received_avp", "$avp(RECEIVED)")
modparam("usrloc", "nat_bflag", FLB_NATB)
#!endif

#!ifdef WITH_TLS
# ----- tls params -----
modparam("tls", "private_key", "/usr/local/etc/kamailio/certs/kamailio.
key")
modparam("tls", "certificate", "/usr/local/etc/kamailio/certs/kamailio.
crt")
modparam("tls", "tls_method", "TLSv1.2+")
#!endif

#!ifdef WITH_ANTIFLOOD
# ----- pike params -----
modparam("pike", "sampling_time_unit", 5)
modparam("pike", "reqs_density_per_unit", 40)
modparam("pike", "remove_latency", 4)

# ----- pipelimit params -----
modparam("pipelimit", "db_url", DBURL)
modparam("pipelimit", "plp_table_name", "pl_pipes")
modparam("pipelimit", "plp_limit_column", "plimit")
modparam("pipelimit", "timer_interval", 60)
modparam("pipelimit", "reply_code", 503)
modparam("pipelimit", "reply_reason", "Limiting")

# ----- htable params -----
modparam("htable", "htable", "ipban=>size=12;autoexpire=600;")
modparam("htable", "htable", "userban=>size=8;autoexpire=900;")
#!endif
```

```
#!ifdef WITH_XMLRPC
# ----- xmlrpc params -----
modparam("xmlrpc", "route", "XMLRPC");
modparam("xmlrpc", "url_match", "^/RPC")
#!endif

#!ifdef WITH_DEBUG
# ----- debugger params -----
modparam("debugger", "cfgtrace", 1)
modparam("debugger", "log_level_name", "exec")
#!endif

# ----- xlog params -----
modparam("xlog", "buf_size", 8192)
modparam("xlog", "log_facility", "LOG_LOCAL1")
modparam("xlog", "prefix", "-xlog: ")

# ----- geoip params -----
modparam("geoip2", "path",
"/usr/local/etc/kamailio/geoip/GeoLite2-City.mmdb")

# ----- secfilter params ------
modparam("secfilter", "db_url", "mysql://kamailio:StrongPassword@localh
ost/kamailio")
modparam("secfilter", "table_name", "secfilter")
modparam("secfilter", "action_col", "action")
modparam("secfilter", "type_col", "type")
modparam("secfilter", "data_col", "data")
modparam("secfilter", "dst_exact_match", 1)

request_route {
    route(REQINIT);
    route(NATDETECT);
    route(CANCEL);
    route(ACK);
    route(WITHINDLG);
    route(AUTH);
    remove_hf("Route");
    route(RECORDR);
    route(REGISTRAR);
    if ($rU==$null) {
```

```
        sl_send_reply("484","Address Incomplete");
        exit;
    }
    route(LOCATION);
}

route[RELAY] {
    if (is_method("INVITE|BYE|SUBSCRIBE|UPDATE")) {
        if(!t_is_set("branch_route"))  t_on_branch("MANAGE_BRANCH");
    }
    if (is_method("INVITE|SUBSCRIBE|UPDATE")) {
        if(!t_is_set("onreply_route")) t_on_reply("MANAGE_REPLY");
    }
    if (is_method("INVITE")) {
        if(!t_is_set("failure_route")) t_on_failure("MANAGE_FAILURE");
    }
    rtpproxy_manage("co");
    if (!t_relay()) {
        sl_reply_error();
    }
    exit;
}

route[REQINIT] {
    set_reply_no_connect();
#!ifdef WITH_ANTIFLOOD
    if(src_ip!=myself) {
        if(!allow_source_address("10") && $sht(ipban=>$si)!=$null) {
            xlog("L_NOTICE","IPBAN -
            Request $rm from blocked user $fu (IP:$si:$sp)\n");
            exit;
        }

        if (!allow_source_address("10") && !pike_check_req()) {
            xlog("L_INFO","IPBAN -
            Blocking $rm from $fu (IP:$si:$sp)\n");
            $sht(ipban=>$si) = 1;
            exit;
        }
    }
```

```
    if(geoip2_match("$si", "src")) {
        if(($gip2(src=>cc))!="CZ") {
            xlog("L_INFO", "Request $rm originating from:
            $gip2(src=>cc)\n");
            send_reply("403", "Forbidden");
            exit;
        }
    }
#!endif
    secf_check_sqli_all();
    if($au =~ "(\=)|(\-\-)|(')|
        (\#)|(\%27)|(\%24)|(\%60)" && $au != $null) {
        xlog("L_INFO","Possible sql injection
        from $rm $fu (IP:$si:$sp)");
        exit;
     }

    if(src_ip!=myself) {
        if(is_method("INVITE|OPTIONS|SUBSCRIBE")) {
            if(!registered("location", "$fu")) {
                sl_send_reply("403", "Forbidden");
                exit;
            }
        }
    }
    secf_check_ua();
    if($? == -2) {
        xlog("L_ALERT", "Request $rm from
        blacklisted user $ua (IP:$si:$sp)\n");
        exit;
    }
    if($sht(userban=>$au::attemps) >= 10) {
        $var(exp) = $Ts - 900;
        if($sht(userban=>$au::last_auth) > $var(exp)) {
            xlog("L_NOTICE", "USERBAN - Request $rm
            from blocked user $fu (IP:$si:$sp)\n");
            sl_send_reply("403", "Try later");
            exit;
        }
        else {
            $sht(userban=>$au::attemps) = 0;
```

```
        }
    }

    if (!mf_process_maxfwd_header("10")) {
        sl_send_reply("483","Too Many Hops");
        exit;
    }
    if(is_method("OPTIONS") && uri==myself && $rU==$null) {
        sl_send_reply("200","Keepalive");
        exit;
    }
    if(!sanity_check("21991", "7")) {
        xlog("Malformed SIP request from $si:$sp\n");
        exit;
    }
}

route[CANCEL] {
    if (is_method("CANCEL")) {
        if (t_check_trans()) {
            route(RELAY);
        }
        exit;
    }
}

route[ACK] {
    if (!is_method("ACK")) {
        if(t_precheck_trans()) {
            t_check_trans();
            exit;
        }
        t_check_trans();
    }
}

route[RECORDR] {
    if (is_method("INVITE|SUBSCRIBE")) {
        record_route();
    }
}
```

```
route[WITHINDLG] {
    if (!has_totag()) return;

# sequential request withing a dialog should
# take the path determined by record-routing
    if (loose_route()) {
        route(DLGURI);
        if (is_method("BYE")) {
            setflag(FLT_ACC); # do accounting ...
            setflag(FLT_ACCFAILED); # ... even if the transaction fails
        } else if ( is_method("ACK") ) {
            # ACK is forwarded statelessly
            route(NATMANAGE);
        } else if ( is_method("NOTIFY") ) {
            # Add Record-Route for in-dialog NOTIFY as per RFC 6665.
            record_route();
        }
        route(RELAY);
        exit;
    }

    if ( is_method("ACK") ) {
        if ( t_check_trans() ) {
            # no loose-route, but stateful ACK;
            # must be an ACK after a 487
            # or e.g. 404 from upstream server
            route(RELAY);
            exit;
        } else {
            # ACK without matching transaction ... ignore and discard
            exit;
        }
    }
    sl_send_reply("404","Not here");
    exit;
}

route[REGISTRAR] {
```

```
    if (!is_method("REGISTER")) return;

    if(isflagset(FLT_NATS)) {
        setbflag(FLB_NATB);
#!ifdef WITH_NATSIPPING
        # do SIP NAT pinging
        setbflag(FLB_NATSIPPING);
#!endif
    }
    if (!save("location")) {
        sl_reply_error();
    }
    exit;
}

route[LOCATION] {

#!ifdef WITH_ASTERISK
    if(is_method("INVITE") && (!route(FROMASTERISK))) {
        route(TOASTERISK);
        exit;
    }
#!endif

    $avp(oexten) = $rU;
    if (!lookup("location")) {
        $var(rc) = $rc;
        t_newtran();
        switch ($var(rc)) {
            case -1:
            case -3:
                send_reply("404", "Not Found");
                exit;
            case -2:
                send_reply("405", "Method Not Allowed");
                exit;
        }
    }

    # when routing via usrloc, log the missed calls also
    if (is_method("INVITE")) {
```

```
        setflag(FLT_ACCMISSED);
    }

    route(RELAY);
    exit;
}

route[AUTH] {

#!ifdef WITH_AUTH
#!ifdef WITH_IPAUTH
    if(!is_method("REGISTER")) {
        # source IP allowed
        return;
    }
#!ifdef WITH_ASTERISK
    if($si==$sel(cfg_get.asterisk.bindip)
        && $sp==$sel(cfg_get.asterisk.bindport)) {
        return;
    }
#!endif
#!endif
    if (is_method("REGISTER") || from_uri==myself) {
        if(!auth_check("$fd", "subscriber", "1")) {
            if($sht(userban=>$au::attemps) == $null) {
                $sht(userban=>$au::attemps) = 0;
            }
            if(is_present_hf("Authorization")) {
                $sht(userban=>$au::attemps) =
                   $sht(userban=>$au::attemps) + 1;
                xlog("L_INFO", "Failed auth - incrementing attemps
                 count for $fu to: $sht(userban=>$au::attemps)");
            }
            if($sht(userban=>$au::attemps) >= 10) {
                xlog("L_WARN", "Too many failed auth in
                 a row from $fu (IP:$si:$sp)\n");
            }
            $sht(userban=>$au::last_auth) = $Ts;
            auth_challenge("$fd", "0");
            exit;
        }
```

```
        $sht(userban=>$au::attemps) = 0;
        if(!is_method("REGISTER|PUBLISH")) {
            consume_credentials();
        }
        xlog("L_INFO", "User $fu authenticated correctly.
        Reseting count: $sht(userban=>$au::attemps) \n");
    }
#!else
    # authentication not enabled -
    # do not relay at all to foreign networks
    if(uri!=myself) {
        sl_send_reply("403","Not relaying");
        exit;
    }
#!endif
    return;
}

route[NATDETECT] {
#!ifdef WITH_NAT
    force_rport();
    if (nat_uac_test("19")) {
        if (is_method("REGISTER")) {
            fix_nated_register();
        } else {
            if(is_first_hop()) {
                set_contact_alias();
            }
        }
        setflag(FLT_NATS);
    }
#!endif
    return;
}

route[NATMANAGE] {
#!ifdef WITH_NAT
    if (is_request()) {
        if(has_totag()) {
            if(check_route_param("nat=yes")) {
                setbflag(FLB_NATB);
```

```
            }
        }
    }
    if (!(isflagset(FLT_NATS) || isbflagset(FLB_NATB))) return;

    if(nat_uac_test("8")) {
        rtpproxy_manage("co");
    } else {
        rtpproxy_manage("cor");
    }

    if (is_request()) {
        if (!has_totag()) {
            if(t_is_branch_route()) {
                add_rr_param(";nat=yes");
            }
        }
    }
    if (is_reply()) {
        if(isbflagset(FLB_NATB)) {
            if(is_first_hop())
                set_contact_alias();
        }
    }

    if(isbflagset(FLB_NATB)) {
        # no connect message in a dialog involving NAT traversal
        if (is_request()) {
            if(has_totag()) {
                set_forward_no_connect();
            }
        }
    }
#!endif
    return;
}

route[DLGURI] {
#!ifdef WITH_NAT
    if(!isdsturiset()) {
        handle_ruri_alias();
```

```
    }
#!endif
    return;
}

branch_route[MANAGE_BRANCH] {
    xdbg("new branch [$T_branch_idx] to $ru\n");
    route(NATMANAGE);
}

reply_route {
    if(!sanity_check("17604", "6")) {
        xlog("Malformed SIP response from $si:$sp\n");
        drop;
    }
}

onreply_route[MANAGE_REPLY] {
    rtpproxy_manage("co");
    xdbg("incoming reply\n");
    if(status=~"[12][0-9][0-9]") {
        route(NATMANAGE);
    }
}

failure_route[MANAGE_FAILURE] {
    rtpproxy_manage("co");
    route(NATMANAGE);

    if (t_is_canceled()) exit;

#!ifdef WITH_BLOCK3XX
    # block call redirect based on 3xx replies.
    if (t_check_status("3[0-9][0-9]")) {
        t_reply("404","Not found");
        exit;
    }
#!endif

#!ifdef WITH_BLOCK401407
    # block call redirect based on 401, 407 replies.
```

```
    if (t_check_status("401|407")) {
        t_reply("404","Not found");
        exit;
    }
#!endif
}

#!ifdef WITH_ASTERISK
route[FROMASTERISK] {
    if($si==$sel(cfg_get.asterisk.bindip)
        && $sp==$sel(cfg_get.asterisk.bindport)) {
        return 1;
    }
    return -1;
}

route[TOASTERISK] {
    $du = "sip:" + $fU + "@" + $sel(cfg_get.asterisk.bindip)
        + ":" + $sel(cfg_get.asterisk.bindport);
    xlog("L_INFO", "Routing to asterisk with destination URI: $du");
    route(RELAY);
    exit;
}
#!endif

event_route[topoh:msg-outgoing] {
    if($sndto(ip) == "10.0.50.55") {
        drop;
    }
}
```

Příloha H: SIPp – G.711A-G.711A

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">
<!--
 field0 - IP address
 field1 - caller name
 field2 - caller authentication
 field3 - callee
-->
<scenario name="uac-invite">

<nop>
  <action>
    <gettimeofday assign_to="seconds,microseconds" />
    <assignstr assign_to="sess_id" value="[$seconds]"/>
    <assignstr assign_to="version_id" value="[$microseconds]"/>
    <ereg regexp="([0-9]+)\." search_in="var" variable="sess_id"
        check_it="true" assign_to="dummy,sess_id_int"/>
    <ereg regexp="([0-9]+)\." search_in="var" variable="version_id"
        check_it="true" assign_to="dummy,version_id_int"/>
  </action>
</nop>

    <send retrans="500" start_rdt="1">
        <![CDATA[

        INVITE sip:[field3]@[remote_ip]:[remote_port] SIP/2.0
        Via: SIP/2.0/[transport] [field0]:5060;branch=[branch]-reg
        From: <sip:[field1]@[field0]:5060>;tag=[pid]SIPp[call_number]
        To: <sip:[field3]@[remote_ip]:[remote_port]>[peer_tag_param]
        Call-ID: [call_id]
        CSeq: [cseq] INVITE
        Contact: <sip:[field1]@[field0]:5060>
        Max-Forwards: 70
        User-Agent: SIPp
        Content-Type: application/sdp
        Content-Length: [len]
        Privacy: none
        P-Preferred-Identity: <sip:[field1]@[field0]:5060>

        v=0
```

```
    o=[field1] [$sess_id_int] [$version_id_int]
    IN IP[local_ip_type] [field0]
    s=Test
    c=IN IP[local_ip_type] [field0]
    t=0 0
    m=audio [media_port] RTP/AVP 8
    a=rtpmap:8 PCMA/8000
    a=ptime:20

    ]]>
</send>

<recv response="100" optional="true" ></recv>
<recv response="180" optional="true" rtd="1"></recv>
<recv response="200" crlf="true" rrs="true">
    <action>
        <ereg regexp="[1-9a-zA-Z].*[^>]"  search_in="hdr"
           header="Contact:" check_it="true" assign_to="1" />
    </action>
</recv>

<send>

    <![CDATA[

    ACK [$1] SIP/2.0
    Via: SIP/2.0/[transport] [field0]:5060;branch=[branch]-reg
    Route: <sip:[remote_ip];lr;ftag=[pid]SIPp[call_number]>
    From: <sip:[field1]@[field0]:5060>;tag=[pid]SIPp[call_number]
    To: <sip:[field3]@[remote_ip]:[remote_port]>[peer_tag_param]
    Call-ID: [call_id]
    CSeq: [cseq] ACK
    Contact: <sip:[field1]@[field0]:5060>
    Max-Forwards: 70
    Content-Length: [len]

    ]]>

</send>

<nop>
```

```
        <action>
            <exec rtp_stream=
            "/home/deployer/scenarios/audio/g711a.wav"/>
        </action>
    </nop>

    <pause milliseconds="60000"/>

    <send retrans="500">
        <![CDATA[

        BYE [$1] SIP/2.0
        Via: SIP/2.0/[transport] [field0]:5060;branch=[branch]-reg
        From: <sip:[field1]@[field0]>;tag=[pid]SIPp[call_number]
        To: <sip:[field3]@[remote_ip]>[peer_tag_param]
        Call-ID: [call_id]
        CSeq: [cseq] BYE
        [routes]
        Contact: <sip:[field1]@[field0]:5060>
        Max-Forwards: 70
        Content-Length: 0

        ]]>
    </send>

    <recv response="200" crlf="true"></recv>
    <ResponseTimeRepartition value=" 50, 100, 200, 300, 400, 500 "/>
    <CallLengthRepartition value="5000, 10000, 30000, 60000"/>
    <Reference variables="dummy"></Reference>
</scenario>
```

Příloha I: SIPp - G.726-G.711A

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">
<scenario name="uac-invite">

<nop>
  <action>
    <gettimeofday assign_to="seconds,microseconds" />
    <assignstr assign_to="sess_id" value="[$seconds]"/>
    <assignstr assign_to="version_id" value="[$microseconds]"/>
    <ereg regexp="([0-9]+)\." search_in="var" variable="sess_id"
        check_it="true" assign_to="dummy,sess_id_int"/>
    <ereg regexp="([0-9]+)\." search_in="var" variable="version_id"
        check_it="true" assign_to="dummy,version_id_int"/>
  </action>
</nop>

    <send retrans="500" start_rdt="1">
        <![CDATA[

        INVITE sip:[field3]@[remote_ip]:[remote_port] SIP/2.0
        Via: SIP/2.0/[transport] [field0]:5060;branch=[branch]-reg
        From: <sip:[field1]@[field0]:5060>;tag=[pid]SIPp[call_number]
        To: <sip:[field3]@[remote_ip]:[remote_port]>
        Call-ID: [call_id]
        CSeq: [cseq] INVITE
        Contact: <sip:[field1]@[field0]:5060>
        Max-Forwards: 70
        User-Agent: SIPp
        Content-Type: application/sdp
        Content-Length: [len]
        Privacy: none
        P-Preferred-Identity: <sip:[field1]@[field0]:5060>

        v=0
        o=[field1] [$sess_id_int] [$version_id_int]
        IN IP[local_ip_type] [fiel
        d0]
        s=Test
```

```
    c=IN IP[local_ip_type] [field0]
    t=0 0
    m=audio [media_port] RTP/AVP 2
    a=rtpmap:2 G726-32/8000
    a=ptime:20

    ]]>
</send>

<recv response="100" optional="true"></recv>
<recv response="180" optional="true"></recv>
<recv response="200" crlf="true" rrs="true" rtd="1">
    <action>
        <ereg regexp="[1-9a-zA-Z].*[^>]"  search_in="hdr"
           header="Contact:" check_it="true" assign_to="1" />
    </action>
</recv>

<send>

    <![CDATA[

    ACK [$1] SIP/2.0
    Via: SIP/2.0/[transport] [field0]:5060;branch=[branch]-reg
    Route: <sip:[remote_ip];lr;ftag=[pid]SIPp[call_number]>
    From: <sip:[field1]@[field0]:5060>;tag=[pid]SIPp[call_number]
    To: <sip:[field3]@[remote_ip]:[remote_port]>[peer_tag_param]
    Call-ID: [call_id]
    CSeq: [cseq] ACK
    Contact: <sip:[field1]@[field0]:5060>
    Max-Forwards: 70
    Content-Length: [len]

    ]]>

</send>

<recv request="INVITE" crlf="true"></recv>
    <nop>
<action>
    <assignstr assign_to="lvia" value="[last_Via:]" />
```

```
    <ereg regexp="[^,]*" search_in="var" variable="lvia"
    assign_to="6" />
    <ereg regexp="[^,]*$" search_in="var" variable="lvia"
    assign_to="7" />
</action>
</nop>

<send>
    <![CDATA[

    SIP/2.0 100 Trying
    [$6]
    Via:[$7]
    [last_From:]
    [last_To:]
    [last_Call-ID:]
    [last_CSeq:]
    Content-Length: 0

    ]]>
</send>

<send>
    <![CDATA[

    SIP/2.0 200 OK
    [$6]
    Via:[$7]
    [last_Record-Route:]
    [last_From:]
    [last_To:]
    [last_Call-ID:]
    [last_CSeq:]
    Contact: <sip:[field1]@[field0]:5060>
    Content-Type: application/sdp
    Content-Length: [len]

    v=0
    o=[field1] [$sess_id_int] [$version_id_int]
    IN IP[local_ip_type] [field0]
    s=Test
```

```
        c=IN IP[local_ip_type] [field0]
        t=0 0
        m=audio [media_port] RTP/AVP 2
        a=rtpmap:2 G726-32/8000
        a=ptime:20

        ]]>
    </send>

    <recv request="ACK" crlf="true"></recv>

    <nop>
        <action>
            <exec play_pcap_audio=
            "/home/deployer/scenarios/audio/g726.pcap"/>
        </action>
    </nop>

    <pause milliseconds="60000"/>

    <send retrans="500" start_rtd="2">
        <![CDATA[

        BYE [$1] SIP/2.0
        Via: SIP/2.0/[transport] [field0]:5060;branch=[branch]-reg
        From: <sip:[field1]@[field0]>;tag=[pid]SIPp[call_number]
        To: <sip:[field3]@[remote_ip]>[peer_tag_param]
        Call-ID: [call_id]
        CSeq: [cseq] BYE
        [routes]
        Contact: <sip:[field1]@[field0]:5060>
        Max-Forwards: 70
        Content-Length: 0

        ]]>
    </send>
    <recv response="200" rtd="2" crlf="true"></recv>
    <ResponseTimeRepartition value=" 50, 100, 200, 300, 400, 500 "/>
    <CallLengthRepartition value="100, 500, 1000, 5000, 10000"/>
    <Reference variables="dummy"></Reference>
</scenario>
```

Příloha J: SIPp – UAS scénář

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">
<scenario name="uas-invite">
    <recv request="INVITE" crlf="true" rrs="true"></recv>

    <nop>
    <action>
        <gettimeofday assign_to="seconds,microseconds" />
        <assignstr assign_to="sess_id" value="[$seconds]"/>
        <assignstr assign_to="version_id" value="[$microseconds]"/>
        <ereg regexp="([0-9]+)\." search_in="var"
            variable="sess_id" check_it="true" assign_to=
        "dummy,sess_id_int"/>
        <ereg regexp="([0-9]+)\." search_in="var"
            variable="version_id" check_it="true" assign_to=
        "dummy,version_id_int"/>
    </action>
    </nop>
    <nop>
    <action>
        <assignstr assign_to="lvia" value="[last_Via:]" />
        <ereg regexp="[^,]*" search_in="var"
        variable="lvia" assign_to="6" />
        <ereg regexp="[^,]*$" search_in="var"
        variable="lvia" assign_to="7" />
    </action>
    </nop>

    <send>
        <![CDATA[

        SIP/2.0 100 Trying
        [$6]
        Via:[$7]
        [last_From:]
        [last_To:];tag=[peer_tag_param]
        [last_Call-ID:]
        [last_CSeq:]
        Contact: <sip:[field0]@[local_ip]:[local_port];
        transport=[transport]>
```

```
    Content-Length: 0

    ]]>
</send>

  <send>
    <![CDATA[

    SIP/2.0 180 Ringing
    [$6]
    Via:[$7]
    [last_Record-Route:]
    [last_From:]
    [last_To:];tag=[peer_tag_param]
    [last_Call-ID:]
    [last_CSeq:]
    Contact: <sip:[field0]@[local_ip]:[local_port];
    transport=[transport]>
    Content-Length: 0

    ]]>
</send>

<send>
    <![CDATA[

    SIP/2.0 200 OK
    [$6]
    Via:[$7]
    [last_Record-Route:]
    [last_From:]
    [last_To:];tag=[peer_tag_param]
    [last_Call-ID:]
    [last_CSeq:]
    Contact: <sip:[field0]@[local_ip]:[local_port]>
    Content-Type: application/sdp
    Content-Length: [len]

    v=0
    o=user1 [$sess_id_int] [$version_id_int]
        IN IP[local_ip_type] [local_ip]
```

```
        s=-
        c=IN IP4 [local_ip]
        t=0 0
        m=audio [auto_media_port] RTP/AVP 8
        a=rtpmap:8 PCMA/8000
        a=ptime:20

        ]]>
    </send>

    <recv request="ACK" crlf="true"></recv>
    <recv request="BYE" rrs="true"></recv>
    <nop>
        <action>
            <assignstr assign_to="ByeVia" value="[last_Via:]" />
            <ereg regexp="[^,]*" search_in="var"
                variable="ByeVia" assign_to="8" />
            <ereg regexp="[^,]*$" search_in="var"
                variable="ByeVia" assign_to="9" />
        </action>
    </nop>
    <send>
        <![CDATA[

        SIP/2.0 200 OK
        [$8]
        Via:[$9]
        [last_Record-Route:]
        [last_From:]
        [last_To:][peer_tag_param]
        [last_Call-ID:]
        [last_CSeq:]
        Contact: <sip:[field0]@[local_ip]:[local_port];
        transport=[transport]>
        Content-Length: 0

        ]]>
    </send>
    <ResponseTimeRepartition value="10, 20, 30, 40, 50, 100, 200 "/>
    <CallLengthRepartition value="100, 500, 1000, 5000, 10000"/>
</scenario>
```

Příloha K: SIPp – Scénář s SQL injekcí

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<scenario name="uac-register">

    <send retrans="500">
        <![CDATA[

        REGISTER sip:[field0] SIP/2.0
        Via: SIP/2.0/[transport] [field0]:[local_port];
        branch=[branch]-reg;rport
        Max-Forwards: 70
        To: <sip:[field1]@[field0]>
        From: <sip:[field1]@[field0]>;tag=[call_number]
        Call-ID: [call_id]
        CSeq: 1 REGISTER
        Contact: <sip:[field1]@[field0]:[local_port]>
        Expires: 3600
        Content-Length: 0
        User-Agent: Sipp

        ]]>
    </send>

    <recv response="100" optional="true">
    </recv>
    <recv response="401" auth="true" rtd="true">
    </recv>

    <send retrans="500">
        <![CDATA[

        REGISTER sip:[field0] SIP/2.0
        Via: SIP/2.0/[transport] [field0]:[local_port];branch=[branch]-
        reg;rport
        Max-Forwards: 70
        To: <sip:[field1]@[field0]>
        From: <sip:[field1]@[field0]>;tag=[call_number]
        Call-ID: [call_id]
        CSeq: 2 REGISTER
        Contact: <sip:[field1]@[field0]:[local_port]>
        Expires: 3600
```

```
        Content-Length: 0
        User-Agent: Sipp
        Authorization: Digest username="1000;UPDATE subscriber
        SET username='hacker' where username='bob'--",
        realm="172.16.32.2"ri="sip:10.0.50.65:5060",
        nonce="XpF6LF6RefDrqObG/C7RbL/DHulKarbEX5/dDYA=",
        response="1171f398209f4de7c8a245a98072f034",alrithm=MD5

        ]]>
    </send>

    <recv response="100" optional="true">
    </recv>

    <recv response="200">
    </recv>

<ResponseTimeRepartition value="100, 200, 300, 400, 500, 1000"/>
<CallLengthRepartition value="5000, 10000, 15000 "/>
</scenario>
```

Příloha L: SIPp – DoS scénář

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE scenario SYSTEM "sipp.dtd">
<scenario name="dos">

<nop>
  <action>
    <gettimeofday assign_to="seconds,microseconds" />
    <assignstr assign_to="sess_id" value="[$seconds]"/>
    <assignstr assign_to="version_id" value="[$microseconds]"/>
    <ereg regexp="([0-9]+)\." search_in="var" variable="sess_id"
    check_it="true" assign_to="dummy,sess_id_int"/>
    <ereg regexp="([0-9]+)\." search_in="var" variable="version_id"
    check_it="true" assign_to="dummy,version_id_int"/>
  </action>
</nop>

    <send retrans="500">
        <![CDATA[

        INVITE sip:dummy@[remote_ip]:[remote_port] SIP/2.0
        Via: SIP/2.0/[transport] [field0]:5060;branch=[branch]-reg
        From: <sip:sipp@[field0]:5060>;tag=[pid]SIPp[call_number]
        To: <sip:dummy@[remote_ip]:[remote_port]>
        Call-ID: [call_id]
        CSeq: [cseq] INVITE
        Contact: <sip:sipp@[field0]:5060>
        Max-Forwards: 70
        User-Agent: SIPp
        Content-Type: application/sdp
        Content-Length: [len]
        Privacy: none
        P-Preferred-Identity: <sip:sipp@[field0]:5060>

        v=0
        o=sipp [$sess_id_int] [$version_id_int]
        IN IP[local_ip_type] [field0]
        s=Test
        c=IN IP[local_ip_type] [field0]
        t=0 0
        m=audio [media_port] RTP/AVP 8
```

```
        a=rtpmap:8 PCMA/8000
        a=ptime:20

        ]]>
    </send>

    <recv response="100" optional="true"></recv>
    <recv response="180" optional="true"></recv>
    <recv response="200" crlf="true">
        <!-- <action>
            <ereg regexp="[1-9a-zA-Z].*[^>]"  search_in="hdr" header=
            "Contact:" check_it="true" assign_to="1" />
        </action> -->
    </recv>

    <ResponseTimeRepartition value="10, 20, 30, 50, 100, 150, 200"/>
    <CallLengthRepartition value="100, 500, 1000, 5000, 10000"/>
    <Reference variables="dummy"></Reference>
</scenario>
```